香港中文大學研究院教育學部
THE CHINESE UNIVERSITY OF HONG KONG
GRADUATE SCHOOL . DIVISION OF EDUCATION

文科教育碩士論文
Master of Arts in Education Thesis

論文題目 Thesis Title	EFFECT OF FLOWCHARTING ON PROGRAM COMPOSITION SKILL 繪畫流程圖對程式編寫能力的影響
撰作語言 Language Used	英文 English
研究生姓名 Name of Student	區世傑 Au Sai Kit
專修範圍 Specialization	教育傳意與科技 Educational Communications and Technology

論文考試委員會
Thesis Examination Committee

論文導師 Thesis Supervisor	Mr. CHUNG Choi Man	鍾財文 先生
校內委員 Internal Examiner	Mr. HAU Kit Tai	侯傑泰 先生
校內委員 Internal Examiner	Dr. LIN Wen Ying	林文鶯 博士
校外委員 External Examiner	Dr. HUNG Sheung Lun	熊尚麟 博士
學部主任 Division Head	Dr. LAM Man Ping	林孟平 博士

論文通過日期
Date of Approval September 4, 1992

EFFECT OF FLOWCHARTING ON PROGRAM COMPOSITION SKILL

by

Au Sai Kit

Under the supervision of

Mr. Chung Choi Man
Mr. Hau Kit Tai
Dr. Lin Wen Ying

A thesis submitted to the
Faculty of Education
The Chinese University of Hong Kong
in partial fulfillment of
the requirements for the degree of
Master of Arts in Education

June, 1992

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Mr. Chung Choi Man, who has given me constant encouragement, guidance, and valuable comments throughout the entire period of this study. It is also my wish to express my thanks to Mr. Hau Kit Tai and Dr. Lin Wen Ying for reading through earlier drafts and giving valuable comments that have helped me make significant improvements.

I would also like to thank Mr. Cheng Ming Leung for his valuable assistance in the marking of the raw scripts. The author would also like to take this opportunity to thank the principals, computer studies teachers and students of the sample schools, for their co-operation and assistance. Without their participation, the study could not have been completed.

My sincere thanks are also given to my wife for both her moral support and assistance in numerous ways throughout the entire period of my research and the preparation of the thesis.

ABSTRACT

Flowcharts are widely used as a program organization tool and are assumed to have positive effects on program development. The purpose of this study was to investigate the effect of flowcharting in developing programs. In the construction of program logic, subjects using BASIC programming language were found to have higher logic scores than subjects using flowchart as organization tool. When flowcharting was not required in program composition, subjects performed better in program logic, semantic correctness and syntactic correctness. Flowcharting seemed to be a redundant task in program development and it did not facilitate the construction of program logic. Male subjects generally performed better than female subjects in some aspects of programming performance. Home computer possession seemed to increase the familiarity with computer and lower the computer anxiety, hence, subjects who possessed home computers generally performed better than those who did not.

Appendices

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

BASIC is the abbreviation for Beginners' All-purpose Symbolic Instruction Codes. It is a language designed for beginners in programming. It is a well-defined subject area which is taught in schools and elsewhere. In Hong Kong, the computer language to be studied in the certificate level is the BASIC language. There are some advantages to use BASIC as programming language in the syllabus. Firstly, most of the micro-computers are installed with BASIC ROMs. The language interpreter is bought together with the computer system. Thus, it is economical to use the language since it does not need extra cost for the language interpreter. Secondly, since the language interpreter is already installed in the computer system, it is very easy to access and to use it. Once the computer system is switched on, the user is already in the BASIC environment.

Due to the limited resources in the secondary schools in Hong Kong, two students have to share one set of microcomputer in the Computer Studies course. Access to computer is limited to certain class periods only. The students are asked to design the algorithm to solve the problem in flowcharts and code the flowcharts into BASIC programs before they go to the computer room. The students are taught with the problem solving procedures(Table 1) to solve complicated problems with

computers, and they are taught to use flowcharts as the organization tools in designing algorithms to solve the problems(CDC Hong Kong, 1986). In the documentation, flowcharts are also required to help the students to comprehend the program during program maintenance.

1.1 Purpose of the research

Flowcharts are the program organization aids used in the Certificate Level Computer Studies in Hong Kong. It is first used between groups of programmers, administrators in different departments of a firm, or between the programmers and the users. Originally, there are a number of versions of flowcharts used in different areas. In 1963, a standard version of flowcharts was proposed by the Sectional Committee operating under the auspices and procedure of the American Standards Association. The set of flowcharts are widely used in different areas and with only a little modification.

Table 1 - Stages of Problem Solving Procedures

Stage	Name of the stage
1	Problem definition
2	Problem analysis
3	Design of an appropriate algorithm
4	Program Coding
5	Program testing and debugging
6	Program Documentation

Program flowcharts show the explicit flow of control and they assist the users to comprehend the programs. On the other hand, flowcharts also prompt the completion of the program logic and they help the programmer to organize the program logic during the composition of programs(McAllister & Brock 1990). In the research of McCormick and Ross(1990), it is found that students performed better in the programming performance test when not required to submit flowcharts. It is explained that flowcharting appears to be regarded by many students as an entirely separate task, rather than as a programming aid. Although flowcharts are commonly used as program design tools, they do not fulfill the requirements of a good program design tool as pointed out by McAllister and Brock (1990).

The effect of flowcharts on programming ability is still a controversial topic. In the teaching syllabus of Computer Studies(CDC Hong Kong , 1986), a lot of time is allocated to flowcharting and programming. Flowcharts are used as the major program design tools and they are also required to show the logic of program in the documentation. It is the time to investigate whether flowcharts perform the required functions.

The purpose of the research is to analyze the effect of flowcharting on the composition of programs.

1.2 Significance of the research

Although there is continual revision on the syllabus of Computer Studies in Hong Kong, the weighting of BASIC programming is still heavy in the syllabus. A good program organization tool is quite beneficial to the students and facilitates them in the acquisition of program composition skills. The result of the research will give insight in the setting up of a better teaching syllabus and may even urge the curriculum development committee to search for a better program organization tool.

CHAPTER 2

LITERATURE REVIEW

2.1 Literature related to cognitive skills in programming

2.1.1 Kolb's Learning Model

Programming is a kind of experiential learning and Kolb's (1971) experiential model of learning is an appropriate model for the activity of programming. The learning model postulated by Kolb is a four-step repetitive cycle as shown in Figure 1.

There are four states in learning through an experience(Collins & White, 1984). These stages are :

1. the concrete experience, which leads to
2. observation and reflections on the experience, which is necessary for
3. formulation of abstract concepts and generalizations, which are then
4. tested in a new situation. This test may then lead to a new concrete experience.

The problem solving procedures(Table 1) is closely related to Kolb's learning model. Figure 2 shows how each of the stages in the problem solving procedures is fitted into the model.

Figure 1 - The Learning Model(Kolb, 1971).

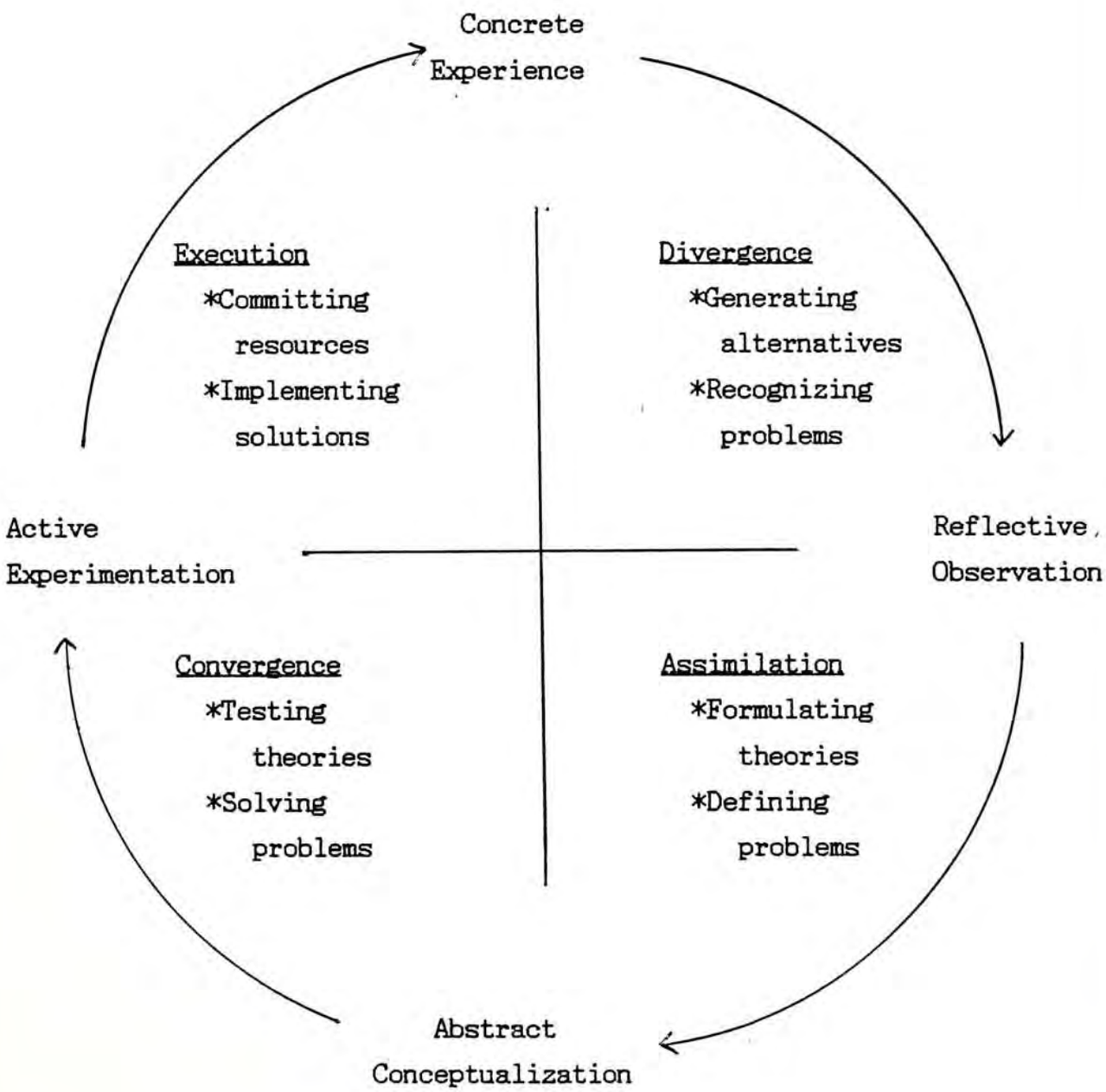
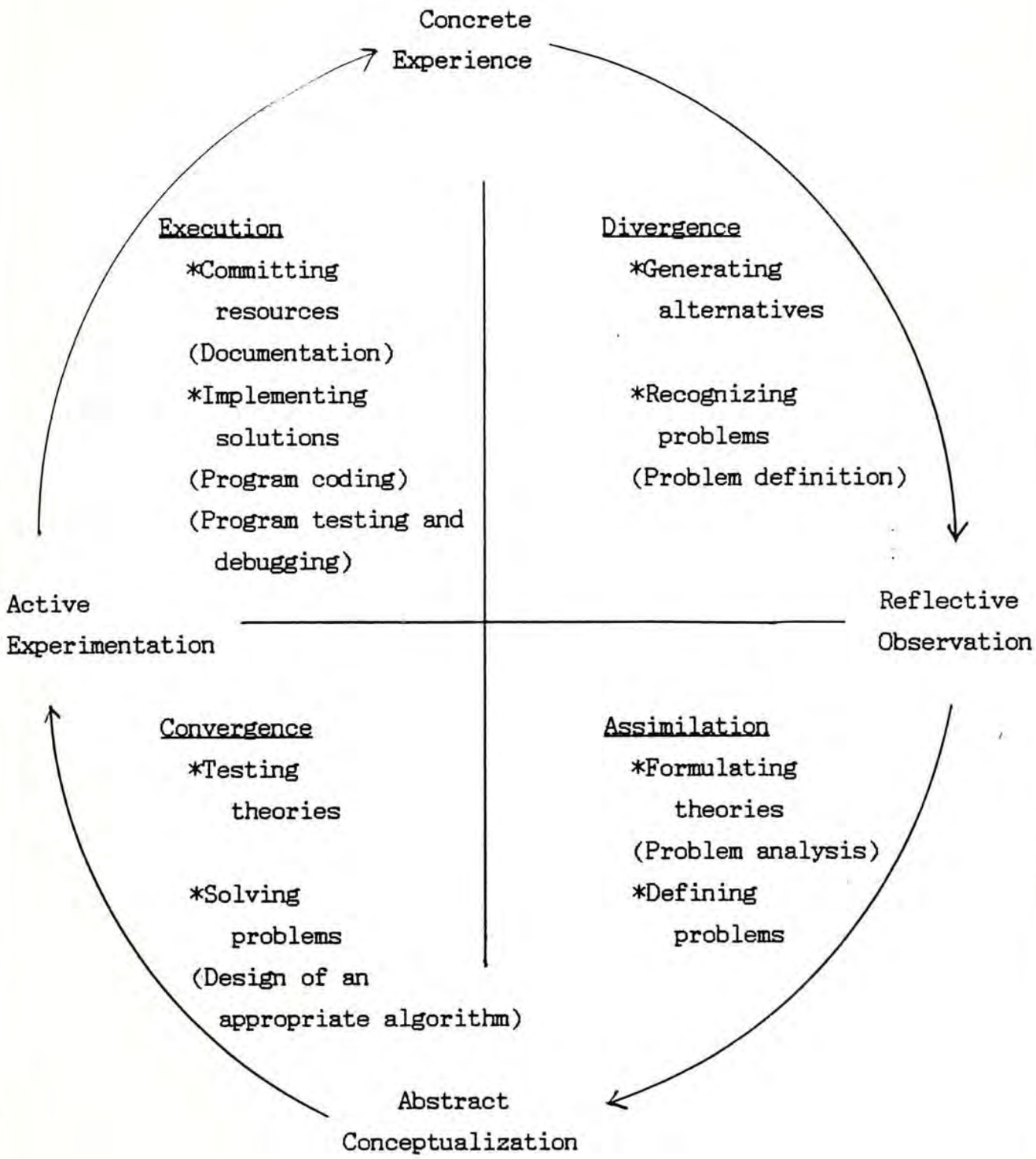


Figure 2 - Diagrams shows how each stage of problem solving procedures is fitted into Kolb's Learning Model.



*Stages of Problem Solving Procedures are in parentheses

2.1.2 The cognitive skills in programming activities

In the view of the progress through the learning stages, there are a number of cognitive skills required in such progression. Shneiderman(1976) identified four relevant cognitive skills involved in such progression:

Comprehension: the ability to understand a written program and the processes occurring as the program executes.

Composition: the ability to analyze a problem and write a program of instructions to solve that problem.

Debugging: the ability to analyze existing code that has errors, locate the errors, and correct them.

Modification: the ability to take a correct program and modify it to achieve another objective.

The relationship of the cognitive skills involved in programming had been investigated (Collins & White, 1984). It is concluded that the ability to correct the error involves creating addition code and is related to the ability to compose a program. Thus the composition skill and the

debugging skill are closely related to each other. It is also found that the comprehension of programs seems to be an easy task for students and is not related to the ability to write programs(Collins & White, 1984).

2.2 Literature related to programming in BASIC

2.2.1 Difficulties in learning BASIC language

Several features of the computer learning can increase the quantity and quality of cognitively demanding activities offered in schools. Firstly, the computer environment is interactive. The computer can respond to the students as soon as they run their programs or supply data to run programs. Secondly, the computer can provide precise feedback. The computer can tell exactly what happen when data is entered in a program. Thirdly, computer learning environments are consistent. The computer can give exactly the same result whenever the same set of data is entered into the same program(Dalbey, Tourniaire, & Linn, 1986).

BASIC language is an interpreted computer language which is most commonly used in secondary schools. Interpreted language can provide a good interactive effect. The students can type in lines of code and run the program in the computer. The interpreter can tell the types and loca-

tion of error once a 'bug' is located. The students can correct the program lines and run the program again until no more error is detected. It is the common practice to most of the students. This type of programming environment encourages the students to program on the computer and this is the common practice of the novice programmers(Dalbey et al., 1986). In this case, the students have not organized their thought before they write the program. They try to solve a programming problem by writing some code and then make small changes in hope of getting it to work. This programming strategy is called 'tinkering'(Perkins, Hancock, Hobbs, Martin, & Simmons, 1986). Sometimes, this strategy can be effective, while at other times it interferes with students' progress in solving programming problems.

Breaking problems down is a general problem-solving strategy. Good ways to break a problem down are of course conditioned by the nature of the programming environment. BASIC programming language does not provide a good programming environment in this aspect(Perkins et al., 1986). Another criticism on BASIC is that GOTO statement is inevitable in a BASIC program. GOTO statement is just a flow control statement. It transfers the control from one part of the program to another and makes the tracing of the program difficult. Usually, a program that contains too many GOTO statements cannot be divided into different functional modules and makes the comprehension of the program

impossible.

2.2.2 Common misconceptions in BASIC language

There are about eight areas of misconceptions commonly found in BASIC programming(Stemler, 1989). In these eight areas of misconceptions, two areas are concerned with logical errors. This kind of logical errors is language independent. Three other types of language-independent conceptual bugs were identified in novice programming(Pea, 1986). The first type of language-independent conceptual bugs is 'parallelism bugs' in which the programmers may have the assumption that different lines in a program can be active or somehow known by the computer at the same time, or in parallel. The second type of language-independent conceptual bugs is 'intentionality bugs' in which the programmers attributed goal directedness or foresightedness to the program. The programmer may glance ahead in the program to see what is to them a familiar programming schema or plan. In both types of bugs, the program has been given the status of an intentional being which has goals, and knows or sees what will happen elsewhere in itself. The third type of language-independent conceptual bugs is 'egocentrism bugs' in which students assume that there is more of their meaning for what they want to accomplish in the program than is actually present in the code they have written. Egocentrism

bugs are the flip side of intentionality bugs. Whereas intentionality bugs involve comprehending and tracing what a program will do, egocentrism bugs are involved in creating a program to do something. Each bug type presupposes that computer can do what it has not been told to do in the program.

2.2.3 The psychology of learning BASIC

There are eight levels of knowledge being identified in BASIC language. The levels of knowledge are machine, transaction, prestatement, statement, mandatory chunk, basic nonmandatory chunk, higher chunk, and program(Mayer, 1979). It is found that expert programmers have a higher ability in the identification of the mandatory chunks, basic nonmandatory chunks, and higher chunks(functional program segment). Since programs are composed of statements, mandatory chunks, basic nonmandatory chunks and higher chunks, the expert programmers should have a higher ability in comprehension of programs. These findings are in good accordance with the research findings of Anderson (1981). In the experiment, it is found that expert programmers recall a very high percentage of the program lines in correct order of a meaningful program, but, recall the same percentage of program lines which are listed in random order. It is believed that the expert programmers store functionally related program lines

in chunks. They have established a schema in which the nodes are functional related. When reading program lines, the nodes are instantiated with the variables and constants. Thus there should be a difference in the programming abilities between the experts and the novices.

One component of expertise is an extensive repertoire of "programming templates." Templates are stereotypic prescriptions for a particular aspect of a program, similar to schemata as described by Norman et al.(1976), or higher chunks as described by Mayer(1979). Research by Kurland and Pea(1984) reveals that expert programmers can articulate their templates, and actively seek new templates. Planning the solutions to programming problems is the ability to determine an appropriate sequence of available templates.

2.3 Literature related to organization aids

2.3.1 Effect of computer access on programming

It is found that students fail to appreciate planning a program(Dalbey, Tourniaire, & Linn, 1986). There are several reason suggested. Firstly, many students can solve most problems without planning. Secondly, students fail to see the benefits of the organization aids for helping them to solve programming problems and they prefer to be on-line

with the computer. Thirdly, students find the on-line experience very motivating and they prefer to be on-line interacting with the computer(Dalbey et al., 1986).

In the study of the effect of computer access on programming performance, for the moderate and higher ability students, it is found that the limited access group has a better performance than the free access group(McCormick & Ross, 1990). In the research, the independent variables are Ability (High, Medium, and Low), Computer-Access(Limited and Unlimited) and Flowcharting(Required and Not-required). The dependent variables are a set of tests on programming performance. Table 2 shows the ability group X computer-access group cell means collapsed across the flowcharting variable.

The ability group main effect is significant in the MANOVA($p < .001$) and in all univariate tests(high > middle > low). As revealed in table 2 the medium-ability and high-ability groups performs better under limited-access than unlimited-access. It is explained that the limited-access group are forced to think and plan before they go to the computers. For the unlimited-access group, the students are urged to key in lines the code when they get the access to a computer. There is no organization stage in this group. The study reveals that an program organization stage is essential for a good programming performance. Similar point is suggested by Dalbey et al.(1986). In the school visit

after a program planning course, it is noticed that almost no one does any formal planning before programming. Typically, students scribble down a few lines of code then go to the computers to fully develop their programs interactively(Dalbey et al., 1986).

Table 2. Subtest Means for Ability x Computer Access Groups (McCormick & Ross, 1990)

	Ability Level and Computer Access					
	Low		Medium		High	
Subtest	UN	LA	UN	LA	UN	LA
(N)	11	13	8	14	16	9
Mental Models(2)	.89	.54	.44	1.12	1.06	1.61
Error Recognition(1)	.48	.32	.34	.64	.73	.85
Interpretation(1)	.50	.34	.50	.63	.74	.83
Programming Problem(2)	.84	.38	.37	.92	1.07	1.53
Programming Templates(2)	.52	.36	.59	.69	1.02	.82
Flowcharting Score (3)	1.36	1.08	1.00	1.93	1.94	2.44
Total (11)	4.59	3.02	3.24	5.93	6.59	8.08

Note : UN = unlimited access; LA = limited access.
Values in parentheses indicate maximum scores

2.3.2 Organization aids for programming

Novice programmers are characterized by a "rush to the computer". They frequently attempt to go from a statement of the program code without any consideration of how to design the code. Novices appear to lack the tools necessary for constructing intermediate states between the problem specification and the problem program(Dalbey, Tourniaire, & Linn, 1986). Expert programmers engage in two complementary techniques on planning solutions to programming problems: top-down design and stepwise refinement(Brooks, 1980). Experts plan their solutions to programming problems by selecting appropriate templates for each problem. Expert programmers can do this because they have a large repertoire of program templates(Perkins, Hancock, Hobbs, Martin, & Simmons, 1986 ; Dalbey et al., 1986). Experts use their knowledge of templates to guide the planning process. For the novices, they may know the advantages of planning before writing the program on the computer. The novices lack the ability to select the appropriate templates in the step-wise refinement of a problem since they have only a limited repertoire of program templates.

To plan or organize a program, the programmers have to use some kinds of program design tools(Daniel et al., 1990) or program organization aids(McCormick & Ross, 1990). There are a number of program design tools used in textbooks.

One of the most commonly used is the structure chart, which shows the overall hierarchy and design. It is difficult to follow the logic flow of the structure chart since the charts alone do not show the dynamics of the logic, such as the change of flow of control. The second popular tool is pseudocode. Pseudocode is concise, easy to manipulate, and shows the hierarchical structure of a program. However, pseudocode is not graphical and is not standardized(Daniel et al., 1990). The third common tool for program design is the flowchart graphic. Flowchart graphic is standardized and reveals the dynamics of program logic. It is easy to follow the control flow in flowcharts. There are a number of other tools such as HIPO Charts, Nassi-Schneiderman Diagrams, Warnier-Orr Diagrams, which are not commonly used (Carey & McLeod, 1988).

Flowcharts use six symbols: parallelograms for input/output, diamonds for decisions, lines with arrows for flow lines, circles for connectors, ovals for start/termination, and rectangles for predefined processes. Flowcharts need a lot of space and a detailed flowchart usually spreads over two or three pages. Hand drawing and redrawing flowcharts can take longer than coding the programs(Daniel et al., 1990). Although flowcharts do not capture all the characteristics of good program design tools, they are the most commonly used design tools in high schools. In Hong Kong, students are taught to use flowchart to represent the

solution of problems. Thus, flowcharting is a very important topic in the secondary schools of Hong Kong.

In the study of the effect of computer access on programming performance, half the subjects in the limited access group is required to submit flowcharts and the other half is not. It is found that the non-flowchart group performed better than the flowchart group(McCormick & Ross, 1990). It is explained that many students do not adequately master flowcharting skills. Flowcharting, because of its pictorial modality and unique symbol system, appears to be regarded as a separate task by most of the students. Some of the students may complete the program first and then draw the flowchart accordingly as to fulfill the requirement of the teachers. It may be concluded that the subjects in the limited access group perform better because they are forced to plan before to key in their programs. The students may have their plan 'organized' abstractly in their mind or in their own set of codes and formats. This finding brings with another question - Does flowcharting really help in the organization of program logic?

2.3.3 Effect of flowcharts on the comprehension of programs

There is consistent evidence that advance organizers have stronger effects for unfamiliar, abstract information

than for familiar, concrete information. Royer and his colleagues(1975) had demonstrated that concrete models may serve as effective advance organizers in learning new scientific information. In their studies, subjects read two passages, such as a passage on electrical conductivity followed by a passage on heat flow. Subjects were asked to read two passages. For some subjects, the first passage contained several concrete analogies, such as electrical conduction being described as a chain of falling dominoes. For other subjects the first passage presented the same information in abstract form without any concrete analogies. It is found that the group showed a better performance in reading of the second passage if concrete analogies are provided in the first passage. In this case the concrete analogies in the first passage serve as the advance organizer for the second passage.

Since a program is made up of discrete program lines. Each program line carries a distinct function. It is necessary to following the logic flow and at the same time, grasp the meaning of the program lines, in order to understand the meaning of the program. One would be easily lost in tracing the program since they only focus on only a small part of program. Thus, to understand a program is quite similar to the process of to understand a abstract paragraph. In the comprehension of a program, the flowchart is quite similar to the effect of an advance organizer. Thus a flowchart

should help in the comprehension of a program(Shneiderman, Mayer, McKay, & Heller, 1977).

A series of experiments was done by Shneiderman and Mayer(1977). The experiments are designed to test whether comprehension can be improved if a detailed flowchart is studied in conjunction with a program. Two forms of the examination were prepared. The first form contained two programs with a flowchart for the first program only, while the second form contained the same two programs but a flowchart for the second program only. An analysis of variance for the scores indicated that there was no significant difference in the performance when a detailed flowchart was provided. It was observed that most subjects were rarely referring to the available flowchart but preferred to study the program directly.

2.3.4 The effect of flowcharting on development of programs

There are a number of organization aids developed for programming. Flowcharts are the most commonly used organization aids since the standardization of flowcharts by the Sectional Committee X3, Computers and Information Processing, operating under the auspices and procedure of the American Standards Association(1963). Flowcharts seem to add direction to the programs of students during the con-

struction of programs. Flowcharting has been considered as an art requiring practice and that a flowchart should be developed before a program is coded. In an experiment on the effect of instruction on the misconceptions in BASIC language, it was observed that the subjects who were required to develop a flowchart, spent less time in developing and debugging a program(Stemler, 1989). It is explained that the use of flowchart design technique forces the student to look more at the solution of the problem in the very beginning stages of software development for the early identification and removal of errors. Conflicting results are found by other researches(McCormick & Ross, 1990; Shneiderman et al., 1977).

In the investigation of effect of flowcharting on programming performance(Table 2. McCormick and Ross, 1990), the MANOVA yields an main effect($p < .05$). It is found that students who are not required to submit flowcharts tend to score higher on all subtests of programming performance than those required to submit them. Because of its modality and unique symbol system, flowcharting appears to be a separate task apart from programming. Many students complete the programs before they write the flowchart accordingly.

In the experimental investigations of the utility of detailed flowcharts in programming(Shneiderman et al., 1977), no statistically significant difference between

flowchart and nonflowchart groups has been shown. Since the basic tasks of programming have been delineated as program composition, comprehension, debugging and modification(Weinberg, 1971), the goal of the experiment is to determine the utility of detailed flowcharts in these computer programming tasks. FORTRAN was the programming language used in the experiment. The subjects were basically the students of some introductory programming course in the universities, thus, they were considered as novice programmers.

The first experiment was designed to study how the creation of a detailed flowchart assisted the subjects in composing a program. The flowchart group was asked to write a flowchart and then a program to a given problem. The non-flowchart group was ask to write a program only. The grading was done by the same marker and was normalized to 100 percent. The mean score of the flowchart group was 94 while that of the nonflowchart group was 95. A t-test showed no significant difference between the two groups. The relatively good scores indicate that all the subjects found the task to be straightforward and a ceiling effect had been imposed onto the result.

Similar results were obtained for experiments on comprehension, debugging and modification of programs. It was pointed out that the type of information obtained from a detailed flowchart and a program appeared to be equivalent.

Thus, detailed flowcharts are merely a redundant presentation of the information contained in the programming language statements. "The flowcharts may even be at a disadvantage because they are not as complete (omitting declarations, statement labels, and input/output formats) and require many more pages than do the concise programming language statements"(Shneiderman et al., 1977).

2.4 Literature related to methodology

2.4.1 Choosing materials

The material selected should be of an appropriate level of difficulty to produce data with desirable statistical characteristics. As demonstrated in the study of Shneiderman and Mayer(1977), scores for the two experimental groups were 94 and 95 respectively, while the maximum possible score was 100. In the study of Luann stemler, it was found that explicit instruction resulted in a reduction in errors on programs that contained a change of flow of control. However, there were no significant differences between the performance of the control group and the experimental group on programs that contained only sequential logic(Stemler, 1989). If the task is too simple to cause an differentiation in the performance, an artificial ceiling effect may be imposed on the results. It was suggested more

difficult logic might be used to cause differentiation in the results. Not only has this probably acted to reduce the magnitude of the experimental difference, but it also invalidates any statistical analyses, such as the t-test, which make the assumption of an underlying normal distribution. Similar results would occur in a study which many of the subjects received scores of zero. Thus the level of difficulty of the programs should be well controlled.

Programs in the 50- to 100-line range are the same size as the modules that constitute the large systems. The results obtained from using these smaller programs are, at least partially, generalizable to larger programs(Brooks, 1980). The size of program in the project work of the students in the certificate level is about the same size. In the research of Dalbey et al. (1986), the program size is about 10 to 50 lines of code and is considered as reasonable. In the HKCEE, the size of program in the program comprehension is about 10 to 50 lines of codes. With the consideration of time, the program size used in this study should be about 10 to 50 lines of codes. With the consideration of difficulties, the program used should include a change of flow of control such as loops, and conditional branching.

2.4.2 Length of instruction in experimental approach

In the experimental investigation of the effect of instruction methods on the programming performance, there is a problem of period of instruction versus effects of intervening factors. If the period of instruction is not long enough, there would be no significant difference in the programming performance(Gasen & Morecroft, 1990; Van Merrienboer, 1990). If the period of instruction is too long, the effect of intervening factors may not be controlled and the internal validity is low. The optimal period of time is very difficult to estimate, therefore, the research design should cater this kind of difficulties.

In this research, the programming performance test is one week after the pretest. The instructions involve only the short briefing session just before each test. Since lengthy instruction is avoided in this research, the effect of intervening factors can be reduced to a minimum.

2.4.3 Choosing subjects

The subjects chosen for a research should be representative for large population and must be relatively uniform in regard to their characteristics and abilities at the point at which they are selected to participate in the

experiment. It is very difficult if the sample is not large and the population is very heterogeneous. For intermediate programming students, even a few months' difference in experience can have a significant effect on performance. The variability of the sample must be controlled. Control of the variability by increasing the sample size is rarely feasible. A better approach is based on assessing the abilities of subjects prior to their participation in the experiment and then either grouping them on the basis of ability(stratification) or adjusting for the initial ability level (covariance analysis). A pretest of programming skills is commonly used to assess subject ability. Besides, the number of subjects used should not be too few as to prevent the subject differences from obscuring the experimental effects(Brooks, 1980).

In this research about 365 Form 5 students were chosen as subjects from 15 secondary schools in Hong Kong. A pretest is given to the subjects to assess the BASIC programming ability. The subjects are then matched in pairs according to the BASIC programming ability.

Certain background variables which were related to programming ability were identified. The literature indicates that ability is related to both computer programming experience (Dalbey & Linn, 1985) and mathematical problem solving(McCoy & Dodl , 1989). Ability is found to have a

strong casual effect on programming ability. Other literature identifies two variables related to computer programming : socioeconomic status (McCoy & Dodl, 1989) and access to a home computer(Dalbey & Linn, 1985). The two variables, gender and socioeconomic status, were found to have a very slight casual effect on programming ability(The values of beta are -0.01 and 0.02 respectively). Thus, the information about the programming experience, access to a home computer are also collected by means of a questionnaire.

CHAPTER 3

METHODOLOGY

3.1 Theoretical framework

As pointed out by Collins and White(1984), programming is a kind of experiential learning and the programming activities fit very well into the Kolb's Learning Model(Figure 2). Formulating theories and solving problems are very important activities in the Learning Model. Similar activity, the formulating of algorithms to solve problems or the organization of the program logic, appears in the problem solving procedure(Table 1) in computer context. The importance of this activity has been revealed by a number of researches. An organization phase may help the student in composing programs(McCormick & Ross, 1990). During the organization phase, the students try to develop steps in logical sequence to solve a problem.

Some kinds of organization aids are required in the constructing of the program logic. Different kinds of organization aids have been developed, but flowchart is the most commonly used organization aid. Flowchart is also the organization aid used in Computer Studies subject in HKCEE. Table 3 shows that 130 periods are allocated to problem solving and BASIC programming. Thus

it is worthwhile to investigate the effect of flowcharts on the programming ability of the Hong Kong students.

The original purpose of flowcharts is to improve the communication between different persons in the same information processing system. Flowcharting is the technique in which symbols are drawn to represent the sequence of operations and the flow of data(Sectional Committee X3, American Standards Association, 1963). The teaching syllabus(Table 3) shows that the students should learn to convert a flowchart into a BASIC program. The role of flowcharts in the problem solving procedure is important. Figure 3 shows the role of flowcharts in the problem solving procedure.

Program coding always means the process of coding a flowchart into a BASIC program. Flowchart is not originally designed for assisting the construction of program logic. Although it is used as the program organization tool in Hong Kong secondary schools, flowchart has a number of disadvantages(Yu, 1989):

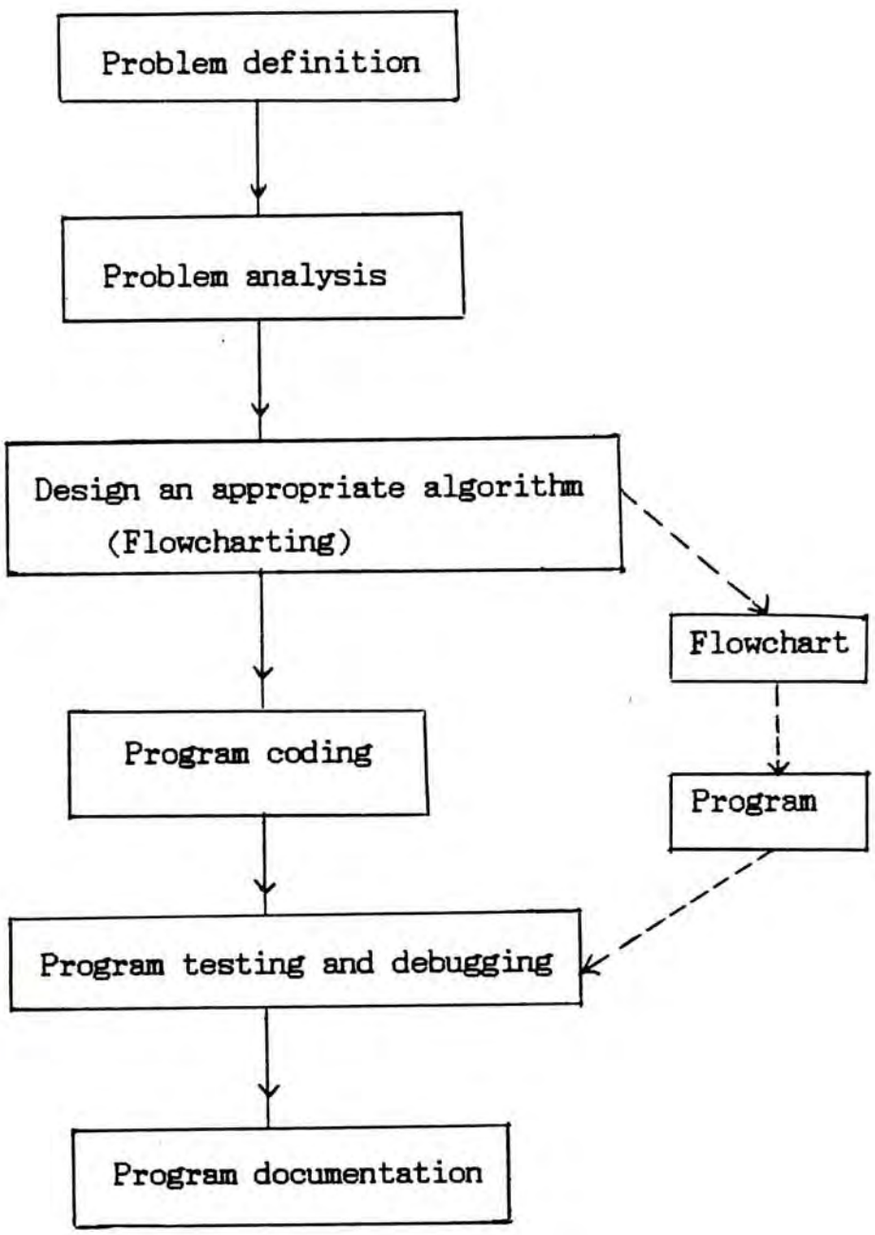
1. Drawing a flowchart requires tedious work and takes much time.
2. Flowcharts are difficult to amend.
3. A flowchart may occupy many pages.
4. The overall structure of the algorithm may not be very obvious unless the flowchart is very carefully drawn.
5. It is difficult to represent complex combinations of conditions by flowcharts.

Table 3 - An estimate allocation of periods in the teaching syllabus(CDC, 1986)

	Topics	Periods
1.	Introductory Computer Concepts	12
2.	Problem Solving and Introduction to Program	
	2.1 Problem solving procedures	2
	2.2 Stepwise refinement as a problem-analysis technique	3
	2.3 Flowcharting as a representation of the algorithm	5
	2.4 Conversion of simple flowcharts into BASIC	3
3.	Programming in BASIC	117
4.	Representation of Information	11
5.	Hardware of a Typical Computer System	13
6.	Data Processing	25
7.	Computer Operation	13
8.	Computers in the Modern World	12
	Total	216

Flowcharting is a time consuming task. Observations in classroom show that flowcharting is considered as a separate task from writing programs. Students always construct the flowcharts accordingly after the completion of the programs. Observations in classroom and the result of previous researches(McCormick & Ross, 1990; Shneiderman, Mayer, McKay, & Heller, 1977) suggest that flowcharting does not assist, sometimes it even hinders the students in the construction of the program logic.

Figure 3 - The role of flowcharts in the problem solving procedure.



Different researches (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986; Anderson, 1981) showed that expert and novice programmers have a different repertoire of codes in their mind. Expert programmers have a well developed schema of programs. They can read related program lines quickly and in larger chunks. While the novices read program in lines as there is no connection between lines. In composition of program, experts break down a problem in certain subtasks and these subtasks instantiate the nodes of their program schema. The experts can write codes in mandatory chunks (a large functional related block of programs) to solve the subtasks effectively. While the novices cannot identify the subtasks of a problem since they do not have a well developed schema of programs. Therefore, there should be a difference in the program comprehension ability and program composition ability between the expert and novice programmers. Since the program schema is language dependent, the comprehension and composition skills in a programming language should depend on the familiarity of the programming language. In Hong Kong, the programming language used in the secondary schools is the BASIC language. Since the knowledge in BASIC may affect the comprehension and composition skills in the language, the knowledge of BASIC of the subject should be measured and taken into account.

3.2 Hypotheses

This research tends to investigate the effect of flowcharting on the program composition skills. As mentioned in the theoretical framework, flowchart is not firstly designed as a program organization tool, besides, flowchart has a number of disadvantages(Yu,1989). It is hypothesized that flowcharting shows no positive effect on program composition skills.

Finally, the null hypotheses addressed in this study are formulated as follows :

1. There is no significant difference in the logic scores between students of FLOWCHART group and BASIC group.

2. There is no significant interaction between program organization tool and ability of subjects in the construction of the logic to solve a problem.

3. There is no significant difference in the logic scores between the subjects of the following dichotomous variables:

   a. gender, and

   b. home computer possession.

4. There is no significant difference in the programming performance(logic score, semantic correctness score and syntactic correctness score) between students of FLOWCHART group

and NON-FLOWCHART group.

5. There is no significant interaction between method of developing programs and ability of subjects in the programming performance(logic score, semantic correctness score and syntactic correctness score).

6. There is no significant difference in the programming performance(logic score, semantic correctness score and syntactic correctness score) between the subjects of the following dichotomous variables :

a. gender, and

b. home computer possession.

3.3 Method

3.3.1 Procedure

A pretest was first given to the subjects to assess their knowledge in BASIC language and flowcharts. Other relevant information was collected in a questionnaire. The experiments were performed one week after the pretest. There were two different experiments applied to two different sets of subjects. In each of the experiments, the subjects were divided into two groups of equal ability according to the pretest scores.

3.3.2 Subjects

The subjects were 355 Form 5 students selected from 15 Anglo-Chinese coeducational schools. Originally 18 schools were chosen at the convenience of the researcher, but due to the tight school schedule of 3 schools, only 15 of them participated in the experiments. All the subjects had studied Computer Studies for about 16 months. 170 subjects were randomly assigned to Experiment I and 185 subjects were randomly assigned to Experiment II. 8 and 11 were absent from Experiment I and Experiment II respectively. These absentees were excluded from the final data analysis.

The subjects in this study were in the age groups 16 to 19, with an average age of 16.71 (S.D. = 0.8853). The age distribution of the subjects is shown Table 4. and the characteristics of the subjects are summarized in Table 5.

3.3.3 Instruments

(i) Questionnaire for students

The subjects were required to respond a questionnaire just before the pretest. Questions were set to collect the following information :

1. Name,
2. sex,
3. age,
4. access to home computer,
5. previous computer course attended,
6. repeat F.5 or not, and
7. attitude towards the use of flowcharts.

Table 4 : Age distribution of subjects.

Age	Experiment I	Experiment II
15	0	2
16	83	85
17	50	56
18	23	24
19	5	5
20	1	2
Total	162	174

Table 5 - Characteristics of subjects

Characteristics		Number in each group	
		Experiment I	Experiment II
Sex	Male	90	112
	Female	72	62
Home computer	Possessed	94	83
	Not possessed	68	91
Repeater	Repeater	6	15
	Non-repeater	156	159
Other Computer course attained	No	139	154
	Yes	23	20

(ii) Test of the knowledge of BASIC language and flowcharts

The test consisted of 30 multiple choice items with 20 on the knowledge of the BASIC programming language and 10 on the knowledge of flowcharts.

The knowledge of the BASIC statements included:

a. input, output, decision, branching, looping, assignment,
b. variables(both numeric and string),
c. arrays(one-dimension and two-dimension only),
d. functions : arithmetic and string functions.
e. subroutines

The basic control structures in BASIC and flowcharts included :
sequence, branching, conditional branching, nested-if,loop, nested-loops, calling subroutine.

(iii) Test for the program composition skill

Test of the construction of program logic

There were two problems in this test. One prob-

lem was low-difficulty and the other one was high-difficulty. In each problem, a problem definition, the format of input data and the required output were given. The subjects were required to write program or flowchart to accept the data and to produce the required output.

Each question carried 10 marks. The marking scheme was modified from that used by Madeo and Bird(1990). The logic scores were given to the logic elements only. Syntax error and misuse in flowchart box deducted no mark. The marking scheme of the questions was set by two experienced markers in the Computer Studies II- HKCEE.

Test of programming composition

There were two problems in this test. One problem was low-difficulty and the other one was high-difficulty. In each problem, a problem definition, the format of input data and the required output were also given. The subjects were required to write program directly or to write program via the process of flowcharting, to accept the data and to produce the required output

The programming performance was assessed in three aspects : the program logic, the program semantics and the program syntax. The marking scheme was modified from that used by Van Merrienboer(1990). The marking of the program composition test was conducted in three steps. Firstly, the number of correctly and incorrectly coded lines were counted. A syntax correctness score was expressed as the percentage of correctly coded lines. Secondly, the logic score was found by counting the number of logic elements present in the program. Finally, the marker scored the semantic correctness of each program on the five-point scale (Table 6) which was modified from that used in the research of Van Merrienboer, 1990.

3.3.4 Design

(i) Definition of terms

a. ABILITY - Ability of subjects
(LOW-ABILITY vs HIGH-ABILITY)

Subjects with pretest scores in the bottom 33.3 percent are considered as low-ability subjects(LOW-ABILITY) and subjects with pretest scores in the top 33.3 percents are considered as high-ability subjects(HIGH-ABILITY).

Table 6 - Scoring sheet for the semantic correctness
(Van Merrienboer, 1990)

Score	Program
0	Not attended.
1	The program is hardly recognizable as a "real" program (e.g., no general input-process-output plan was used).
2	The program can be recognized as a "real" program but obviously does not try to reach its goal because the functional units do not perform the required task.
3	The program clearly tries to reach its goal but includes both semantical and syntactical errors.
4	The program is semantically correct but includes syntactical errors.
5	The program is semantically as well as syntactically correct.

b. METHOD - Method to write program
(FLOWCHART vs NON-FLOWCHART)

The subjects in the FLOWCHART group write programs via the process of flowcharting while the subjects in the NON-FLOWCHART group write programs without flowcharting.

c. TOOL - Tool to construct the program logic
(BASIC vs FLOWCHART)

The subjects in the FLOWCHART group con-

struct the logic by drawing flowcharts while the subjects in the BASIC group construct the logic by BASIC language.

d. HOMECPU - Home computer possession (CPU vs NOCPU)

Subjects who possess home computer are considered as the CPU group while subjects who do not possess home computer are considered as the NOCPU group.

e. Low-difficulty problem - The algorithm to solve the low-difficulty problem requires decisions and one single loop only.

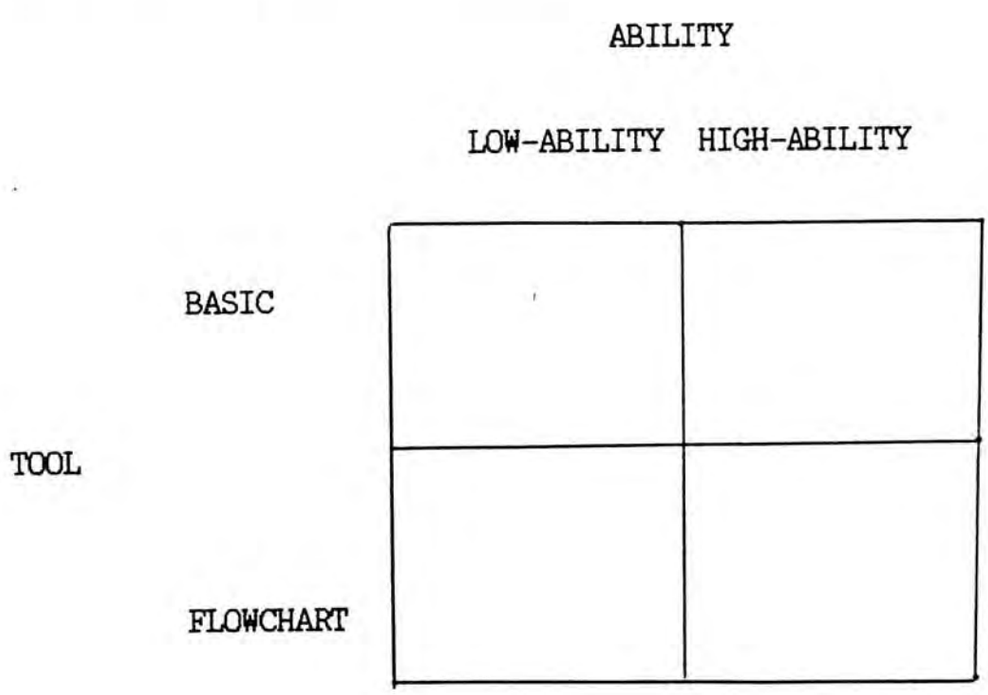
f. High-difficulty problem - The algorithm to solve the high-difficulty problem requires decisions and a nested loop.

g. Programming performance - The programming performance is assessed in three aspect : program logic, program semantics and program syntax.

(ii) Experiment I - Effect of tools on the construction of program logic

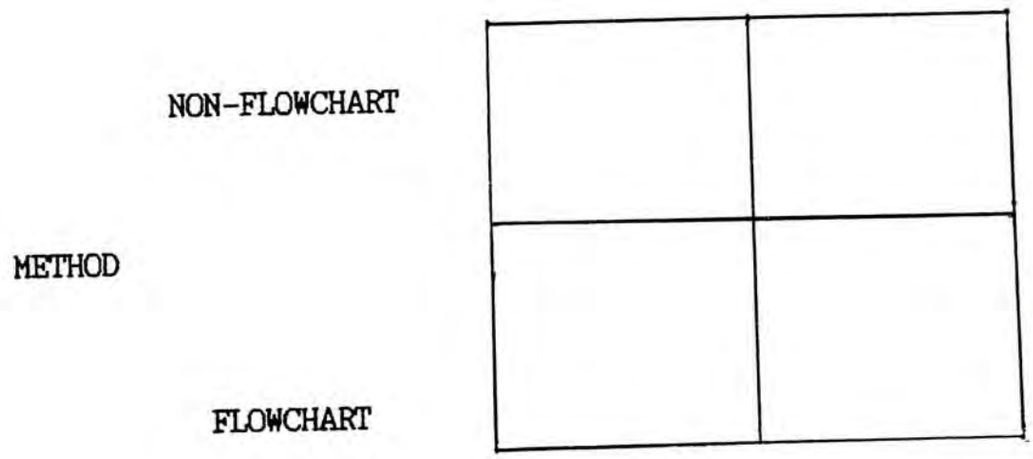
The study is a 2 X 2 factorial design (Figure 4) with tools used to construct the program logic (FLOWCHART vs BASIC) and abilities of subjects (LOW-ABILITY vs HIGH-ABILITY) as between group factors.

Figure 4 - A 2 X 2 factorial design for Experiment I.



Each cell contained the logic scores.

Figure 5 - A 2 X 2 factorial design for Experiment II.



Each cell contained the syntax correctness scores, the logic scores and the semantic correctness scores.

(iii) Experiment II - Effect of flowcharting on program composition

The study is a 2 X 2 factorial design (Figure 5) with methods of writing program(FLOWCHART vs NON-FLOWCHART), and abilities of subjects (LOW-ABILITY vs HIGH-ABILITY) as between group factors.

3.3.5 Analysis

(i) Effect of flowcharting on the construction of program logic

The independent variables were ABILITY (LOW-ABILITY vs HIGH-ABILITY) and TOOL(FLOWCHART vs BASIC). The dependent variables were the logic score of low-difficulty problem(LOGICA1) and the logic score of the high-difficulty problem(LOGICA2).

An ANOVA was applied to the data to see any significant difference between groups and any significant interaction across groups. The basic ANOVA(Figure 4) was a two-way factorial consisting of 2(ABILITY) X 2(TOOL).

An ANCOVA with the pretest scores as the covariate was applied to the logic scores to see any significant difference between male students and female students, and also between students possessing home computer and students not possessing home computer.

(ii) Effect of flowcharting on program composition

The independent variables were ABILITY (LOW-ABILITY vs HIGH-ABILITY) and METHOD(FLOWCHART vs NON-FLOWCHART). The

dependent variables were the syntax correctness score, the logic score and the semantic correctness score(SYNCORR1, LOGICA1 and SMNTA1) of the low-difficulty problem, and the syntax correctness score, the logic score and the semantic correctness score(SYNCORR2, LOGICA2 and SMNTA2) the high-difficulty problem.

An ANOVA was applied to the data to see any significant difference between groups and any significant interactions across groups. The basic ANOVA(Figure 5) was a two-way factorial consisting of 2(ABILITY) X 2(METHOD).

An ANCOVA with the pretest scores as the covariate was applied to the logic scores, the semantic correctness scores and the syntactic correctness scores, to see any significant difference between male students and female students, and also between students with home computer and students without home computer.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Reliability of the instruments

4.1.1 Pretest

A pretest(Appendix 6) was employed to test the knowledge of the BASIC Language and flowcharts. There were altogether 30 multiple choice items. In the pilot test, it was observed that most of the students completed the test in about 35 minutes. The Cronbach alpha reliability coefficient of the pilot test was .7778. In the pretest, the subjects were asked to respond a questionnaire and do the multiple choice items in a double lesson of about 70 minutes. In the pretest of the experiment, 45 minutes was given to the subjects as to ensure that there was ample time for the subjects to complete the test.

There were totally 336 subjects taking the pretest. The mean score and standard deviation of the pretest were 21.97 and 5.69 respectively. The Cronbach alpha reliability coefficient of the pretest was 0.8658(N = 336) which was quite substantially high.

4.1.2 Questionnaire

The questionnaire(Appendix 13) served to collect the relevant information of the subjects. Four questions(questions 8 to 11) were set to measure the attitude of the subjects towards the use of flowcharts in developing programs. The subjects responded to these four questions in a five-point Likert scale. The results were summarized in Table 15. The Cronbach alpha reliability coefficient for these four questions was 0.483(N = 336). The reliability was relatively low. This might be caused by the small number of items and a large number of subjects.

4.1.3 Test of construction of program logic

A logic score was given to the logic of each problem according to the scoring sheet(Appendix 11). Scoring was conducted by the researcher, and about one fifth of the raw scripts were photocopied and marked by another experienced computer studies teacher who was unfamiliar with the experiment. Interrater reliability for the low-difficulty problem and high-difficulty problem were 0.9441 and 0.9554 respectively.

4.1.4 Test of program composition

The scoring for the programming performance was carried out in three stages. For each of the problems, the logic elements were counted as the logic score according to the scoring sheet (Appendix 12). A semantic correctness score was then given according to the marking scheme shown in Table 6. Lastly, the number of correctly coded lines and wrongly coded lines were counted. A syntactic correctness score was calculated as the percentage of lines correctly coded.

The marking of scripts were conducted by the researcher. About one fifth of the raw scripts were photocopied and marked by the same rater in experiment one. The rater gave only the logic score and semantic correctness score for each of the problems, since these two scores were more subjective in the marking system. The interrater reliability coefficients were shown in Table 7. As seen in the table, the interrater reliabilities of the logic score and the semantic correctness scores were found extremely high, thus, the internal consistency of the tests was definitely enhanced.

Table 7 - The interrater reliabilities

	Score	Reliability coefficient
Experiment I (N = 33)	LOGICA1	0.9441
	LOGICA2	0.9554
Experiment II(N = 41)	LOGICA1	0.9391
	LOGICA2	0.9309
	SMNTA1	0.8640
	SMNTA2	0.9304

4.2 Results and discussion

4.2.1 Experiment I

Test for ability in construction of program logic

The first null hypothesis

There were two problems in experiment one. The first problem was a low-difficulty problem which contained only decisions and a single loop structure while the second problem was a high-difficulty problem which contained decisions and a nested loop structure. Each of the problems carried ten points. Thus the logic score ranged from 0 to 10. The mean and standard deviation of the logic score for the first problem was 6.8395(N = 162) and 3.2263 respectively. The mean and standard deviation of the logic score for

the second problem was 6.3951(N = 162) and 3.5898 respectively.

The mean score and the standard deviation for the low-difficulty problem were 7.527(N = 55) and 3.611 for the BASIC group, and 5.796(N = 54) and 3.652 for the FLOWCHART group respectively. An analysis of variance for the logic scores($F(1,105) = 11.059$, $p < 0.001$) indicated that there was a significant difference between the BASIC group and the FLOWCHART group for the logic scores at the 0.001 confidence level. The results showed that the BASIC group performed significantly better in the construction of the program logic for the low-difficulty problem. Thus, the first null hypothesis was rejected for the low-difficulty problem.

The mean score and the standard deviation for the high-difficulty problem were 7.382(N = 55) and 3.424 for the BASIC group, and 5.667(N = 54) and 3.608 for the FLOWCHART group respectively. The result($F(1,105) = 9.285$, $p < 0.01$) indicated that there was a significant difference between the two groups (BASIC and FLOWCHART) for the logic scores of the high-difficulty problem. Therefore, the first null hypothesis was rejected for the high-difficulty problem.

The results suggested that students constructed program logic better when the programming language itself was used, and flowcharts seemed to have a less effect on the

construction of program logic. It was found that the programming language was more precise than the flowcharts in presenting some programming task(Ramsey, Atwood, & Doren,1983). Students might found it easier to use the programming language to present the logic to solve problems.

It was found that students solved programming problems by matching suitable program templates with the subtask of the program(Kurland & Pea, 1984; Dalbey, Tourniaire, & Linn, 1986). Thus the programming ability depended much on the repertoire of templates in the long term memory. If no suitable template was found, students might try to figure out the logic in their mind and write down lines of codes directly(McCormick & Ross, 1990). Thus the students usually by-passed the stage of flowcharting in the development of programs. Flowcharting was treated as an entirely separate task in programming (McCormick & Ross, 1990) and this was the usual practice of the students in Hong Kong. When the students were forced to use flowcharts to construct program logic, they might find it difficult. This might hinder the construction of program logic in the development of programs.

The second null hypothesis

The interaction effects for logic score of the low-

difficulty problem between the ability of subjects and the tool was shown in Figure 6. The results showed that there was an interaction effect for the logic score of the low-difficulty problem(2-way interaction of $F(1,105) = 3.792$, $p < 0.05$). Figure 7 showed that there was no significant interaction effect (2-way interaction of $F(1,105) = 0.819$, $p > 0.05$) for the logic score of the high-difficulty problem. Thus, the second null hypothesis was rejected for the low-difficulty problem but not for the high-difficulty problem.

Figure 6 - Interaction of tool and ability of students (Logic score of low-difficulty problem - Experiment I).

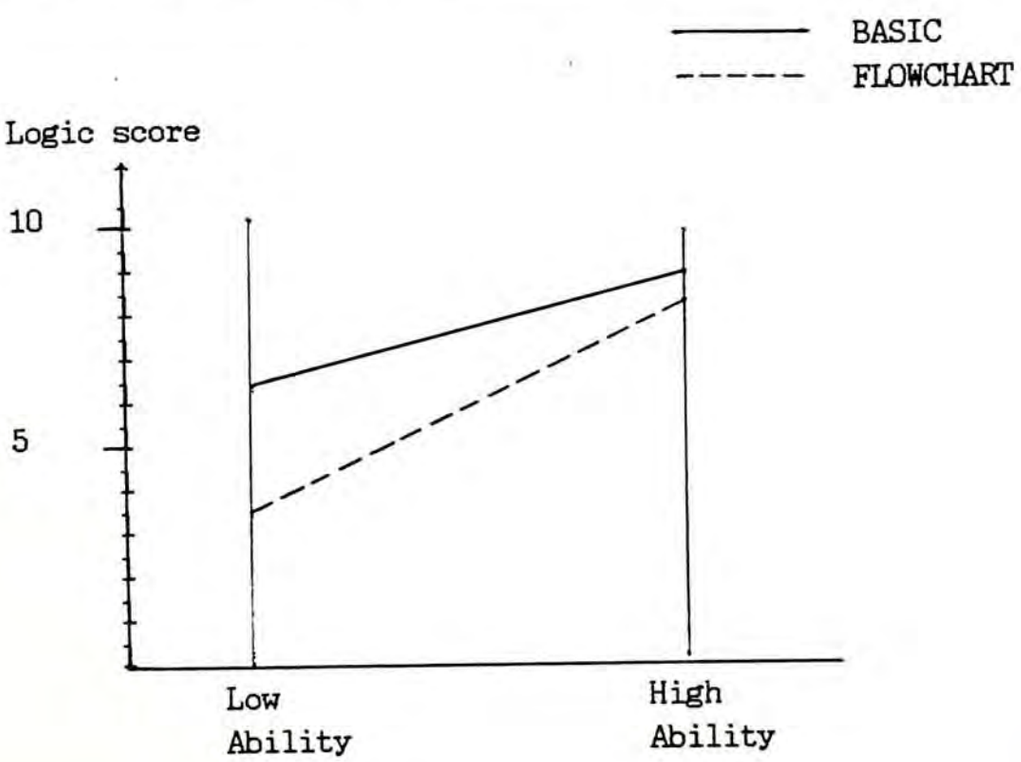
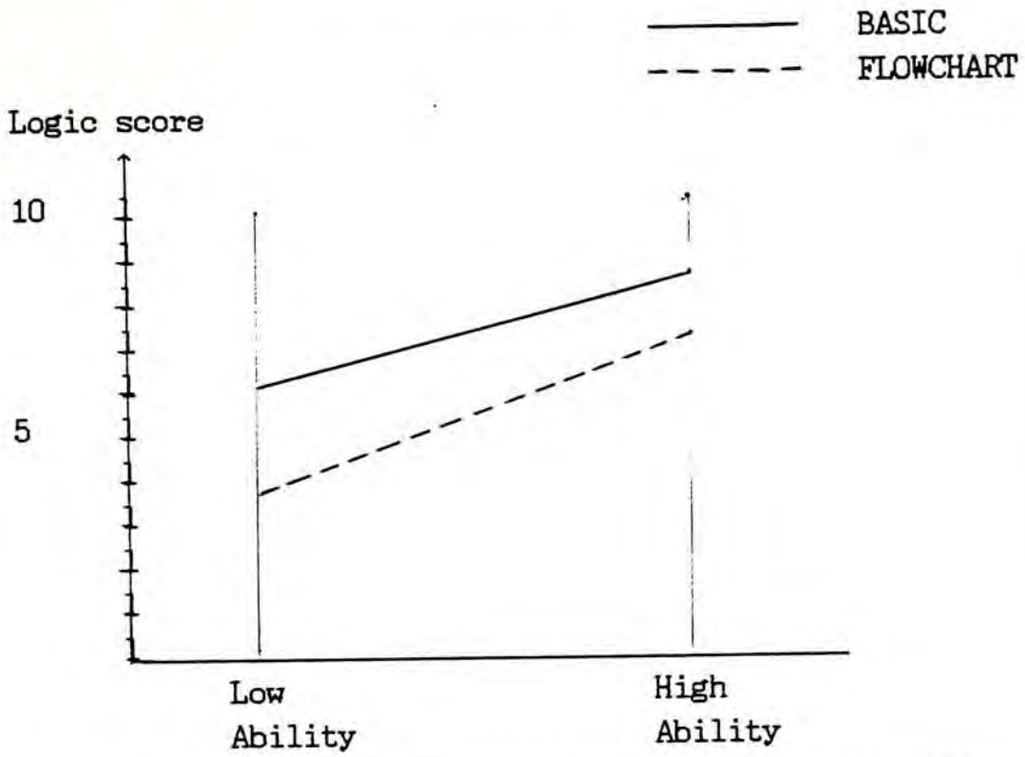


Figure 7 - Interaction of tool and ability of students (Logic score of high-difficulty problem - Experiment I).



The results of interaction effects showed that the effect of tool on the construction of program logic might be different for high and low ability of students. The effect was more obvious for the low-difficulty problem. The result indicated that the effect of tool was more important for the low-ability group. It was explained that the low-ability depended more on the use of tool. Novice programmer tended to rush to the computer, without a good planning before keying lines of code into the computer(McCormick & Ross, 1990). Thus, the low-ability students tended not to plan the solution before writing programs. Therefore, most of the low-ability students did not draw flowcharts before writing programs and they did not use flowcharts in the

construction of logic effectively.

For the high-ability students, they tended to plan before writing programs(Dalbey, Tourniaire, & Linn, 1986; McCormick & Ross, 1990). Some of them appreciated the function of flowcharting and they draw flowcharts before writing programs. Thus, some of the high-ability students might have a similar ability in drawing flowcharts with that in writing programs. On the other hand, most of the high ability students might solve the problem by searching suitable program templates in their large repertoire in the long term memory. The high-ability students drew the flowcharts according to the templates searched. Thus, the high-ability students worked in the reversed way as revealed by other researches(McCormick & Ross, 1990;Dalbey, Tourniaire, & Linn, 1986). This practice enhanced the performance of construction of program logic by flowcharts, therefore, the high-ability students showed small difference in the construction of program logic by flowcharts and by BASIC language.

A better tool facilitated the organization of logic to solve problems. Therefore, the BASIC group performed better than the FLOWCHART group, but the difference in the high-ability students was not so obvious as that in the low-ability students.

The third null hypothesis

As seen in Table 8, the analysis of covariance of the logic scores of the low-difficulty problem($F(1,159) = 6.618$, $p < 0.01$) was significant while that of the high-difficulty problem($F(1,159) = 0.758$, $p > 0.05$) was not significant. The male students had better performance in the construction of logic and the effect was statistically significant for the low-difficulty problem. It might be explained that the male students generally had a more logical mind than the female students, and this was reflected in the better performance of male students in mathematics and programming(McCoy & Dodl, 1989; Hall & Cooper, 1991).Thus, the male students should have higher logic scores. In this case, the low-difficulty problem seemed to have a high power to distinguish the male and female students.

Table 8 - An analysis of covariance of the logic scores by SEX with pretest score as covariate

Logic Score	Mean of Male (N = 90)	Mean of Female (N = 72)	df	F
LOGICA1	7.63(2.82)	5.85(3.44)	1/159	6.618**
LOGICA2	6.86(3.69)	5.82(3.40)	1/159	0.758n.s.

** p < 0.01

n.s. - non-significant

The results in Table 9 showed that both the analysis of covariance of the low-difficulty problem($F(1,159) = 0.297$, $p > 0.05$) and the high-difficulty problem($F(1,159) = 3.490$, $p > 0.05$) were not significant between the NOCPU group and the CPU group. Thus the third null hypothesis was not rejected for the NOCPU group and the CPU group.

As seen in Table 9, the subjects who possessed home computer mainly engaged in the activities such as word-processing, games and investigation of other software. About one third of subjects used the home computer for programming activities. Thus, the students who possessed home computers may have more practice in construction of program logic. Students with home computer could get access to the computer freely. They were more familiar with the computer environment and had a low computer anxiety. Since computer anxiety played an important role in student success in computer achievement, thus, these students should have a better programming performance(Marcoulides, 1988; Loyd & Gressard, 1984). Therefore, students who possessed home computer should have a higher logic score, as shown in LOGICA1(Table 9), but the difference was not significant in this study.

Table 9 - An analysis of covariance of logic scores(LOGICA1 and LOGICA2) by Home computer possession with pretest score as covariate

Logic Score	Mean of NOCPU group (N = 94)	Mean of CPU group (N = 68)	df	F
LOGICA1	6.54(3.38)	7.25(2.98)	1/159	0.297n.s.
LOGICA2	6.46(3.42)	6.31(3.83)	1/159	3.490n.s.

* p < 0.05; ** p < 0.01; *** p < 0.001

n.s. - non-significant

Standard deviation enclosed in brackets.

Table 10 - The activities engaged in home computers

Activity	Frequency
Playing computer game	117
Word Processing	54
Programming	107
Others(Spread sheet, Data processing)	25

4.2.2 EXPERIMENT II

Test for the programming ability

There were also two problems in Experiment II. The first problem was a low-difficulty problem which contained only decisions and a single loop structure while the second

problem was a high-difficulty problem which contained decisions and a nested loop structure. Three different scores(logic score, semantic correctness score, and the syntactic correctness score) were given for each of the problem. The problem set and the marking scheme were shown in appendix 10 and 12 respectively. The logic score ranged from 0 to 10, the semantic correctness score ranged from 0 to 5, and the syntactic correctness score which was a percentage of correctly coded lines, ranged from 0 to 100. The mean and standard deviation of the scores for the complete sample and different groups were shown in Table 11.

The fourth null hypothesis

Generally speaking, the NON-FLOWCHART group performed better than the FLOWCHART group in all the three scores(logic score, semantic correctness score, and syntactic correctness score) in programming performance(Table 11). In the analysis of variance between the NON-FLOWCHART group and the FLOWCHART group, significant differences were only found in the logic scores($F(1,123) = 4.151$, $p < 0.05$) and semantic correctness scores($F(1,123) = 5.171$, $p < 0.05$) of the high-difficulty problem.

For the high-difficulty problem, the NON-FLOWCHART group performed better than the FLOWCHART group in the logic

scores and the semantic correctness scores. This indicated that flowcharting did not really facilitate the construction of program logic. The students in the FLOWCHART group had to construct the flowcharts before writing the programs. Since most of the students did not handle flowchart well(Ramsey, Atwood, & Doren, 1983; Darbey, Tourniaire, & Linn, 1986), as revealed in Experiment I, this hindered the students in the construction of program logic and this was reflected in the logic scores. Difficulties might be found when the program was written according to the flowchart since the logic presented in the flowcharts was not complete. This affected the overall semantics of the program and caused a low score in the semantic correctness.

Table 11 - Analysis of variance by METHOD and ABILITY

Variable	NON-FLOWCHART (N = 87)	FLOWCHART (N = 87)	df	F
LOGICA1	6.632(3.024)	6.437(2.692)	1/123	0.031 n.s.
LOGICA2	3.333(4.195)	2.0811(2.211)	1/123	4.151 *
SMNTA1	3.356(1.257)	3.207(1.183)	1/123	0.358 n.s.
SMNTA2	1.839(1.354)	1.276(1.148)	1/123	5.171 *
SYNCORR1	91.157(15.731)	88.560(20.452)	1/123	0.282 n.s.
SYNCORR2	76.187(36.678)	62.739(41.611)	1/123	2.856 n.s.

* $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

n.s. - non-significant

Standard deviation enclosed in brackets.

There was no significant difference in the syntactic correctness scores between the two groups for both the low-difficulty($F(1,123) = 0.282$, $p > 0.05$) and high-difficulty problems($F(1,123) = 2.856$, $p > 0.05$). The syntactic correctness of a program depended on the knowledge of BASIC of the students. Since the two groups were matched with the pretest scores, both groups should have the same level of knowledge in BASIC language. Therefore, there should be no significant difference in the syntactic correctness scores between the two groups.

Finally, the fourth null hypothesis was only rejected for the logic scores and semantic correctness scores of the high-difficulty problem.

Table 12 - Two-way interactions in programming performance between METHOD and ABILITY

Variable	df	F
LOGICA1	1/123	2.387 n.s.
LOGICA2	1/123	0.356 n.s.
SMNTA1	1/123	0.413 n.s.
SMNTA2	1/123	3.228 n.s.
SYNCORR1	1/123	0.155 n.s.
SYNCORR2	1/123	2.900 n.s.

n.s. - non-significant

The fifth null hypothesis

The results(Figure 8 to Figure 13) showed that no significant interaction effect between groups (METHOD x ABILITY) was found in the programming performance(the logic score, the semantic correctness score and the syntactic correctness score) of both the low-difficulty and high-difficulty problems(Table 12). Therefore, the fifth null hypothesis was not rejected.

Figure 8 - Interaction of method of developing programs and ability of students (Logic score of low-difficulty problem - Experiment II).

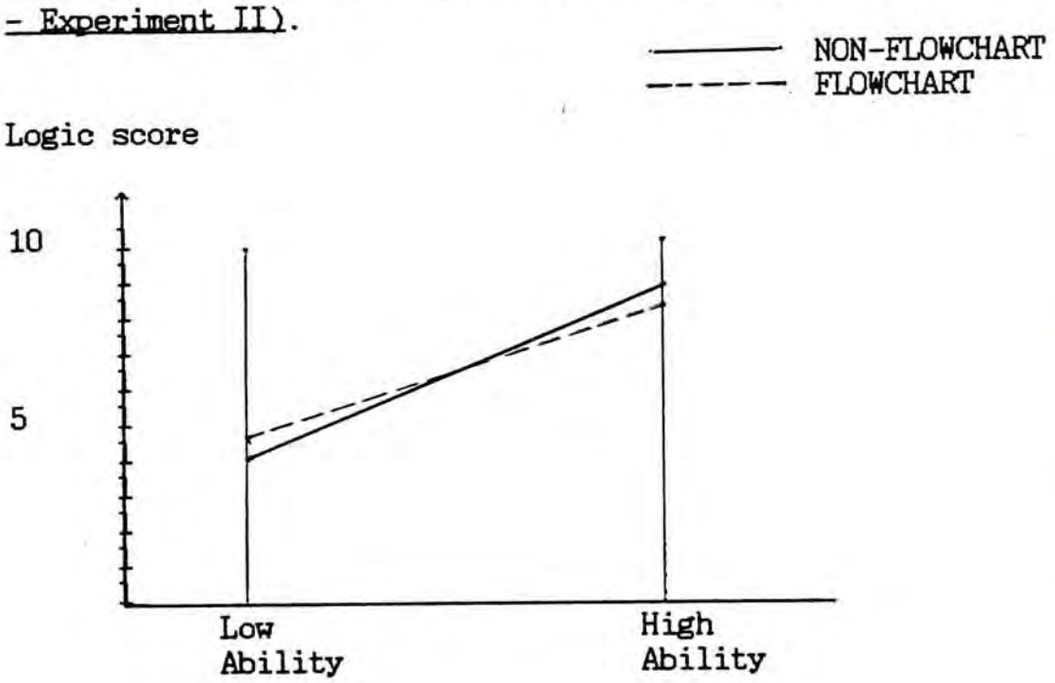


Figure 9 - Interaction of method of developing programs and ability of students(Logic score of high-difficulty problem - Experiment II).

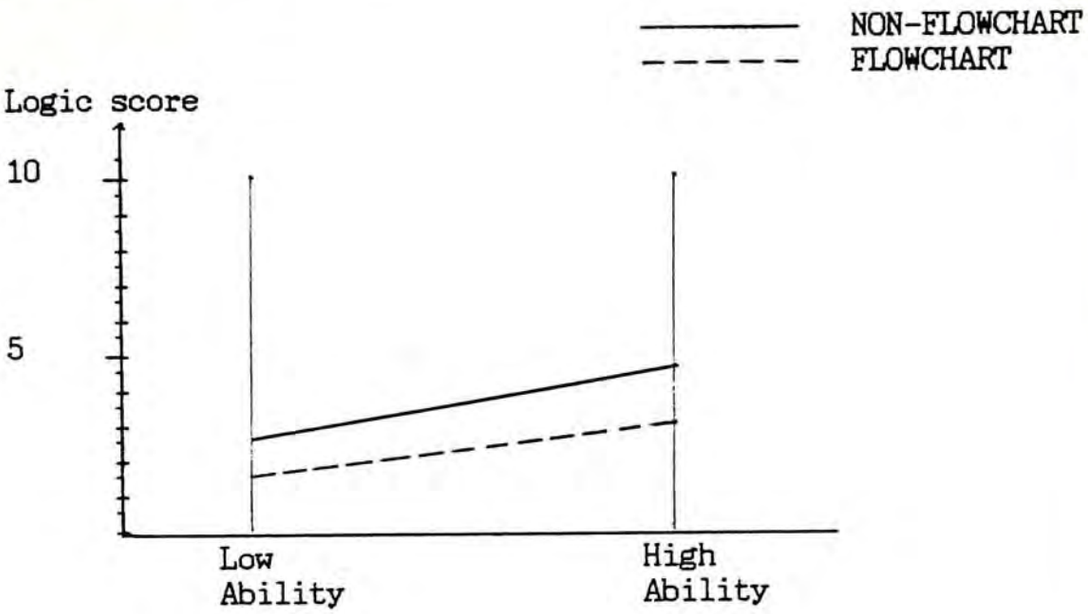


Figure 10 - Interaction of method of developing programs and ability of students (Semantic correctness score of low-difficulty problem - Experiment II).

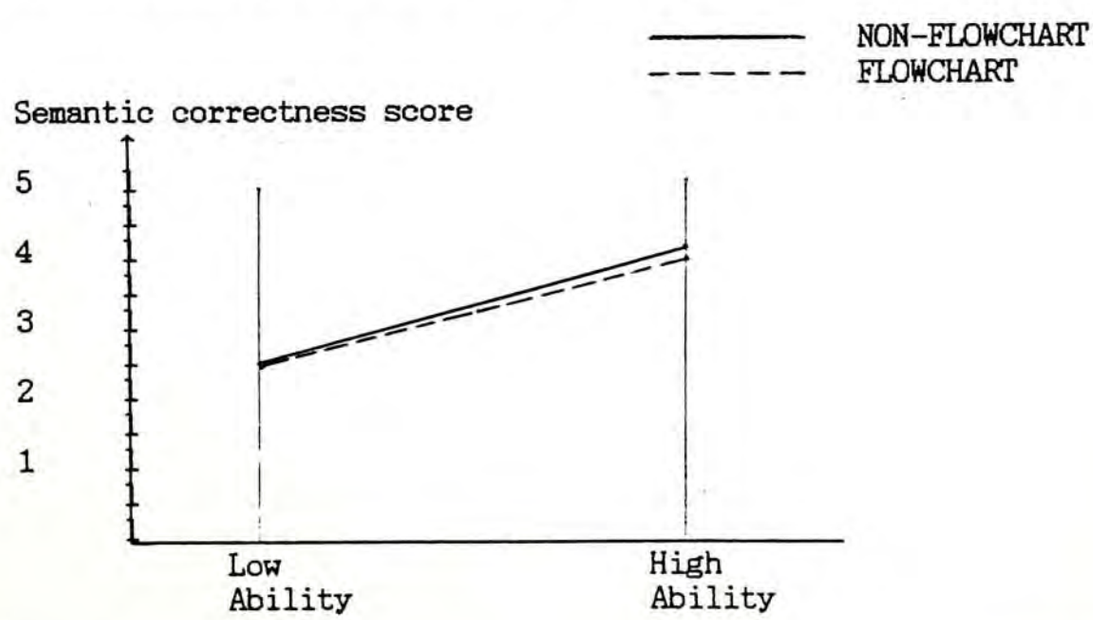


Figure 11 - Interaction of method of developing programs and ability of students (Semantic correctness score of high-difficulty problem - Experiment II).

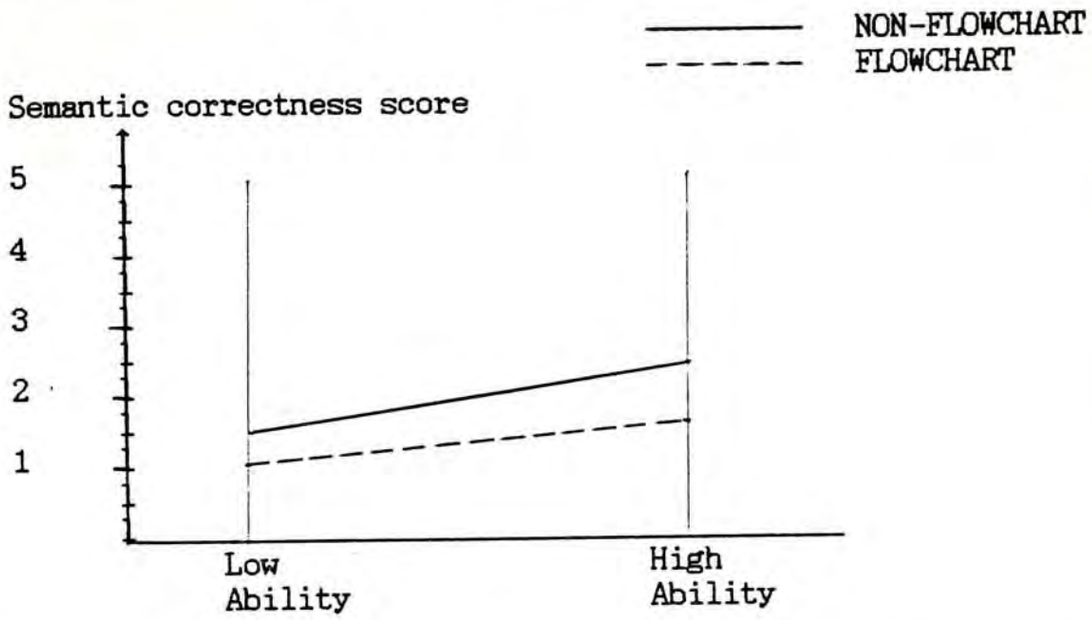


Figure 12 - Interaction of method of developing programs and ability of students (Syntactic correctness score of low-difficulty problem - Experiment II).

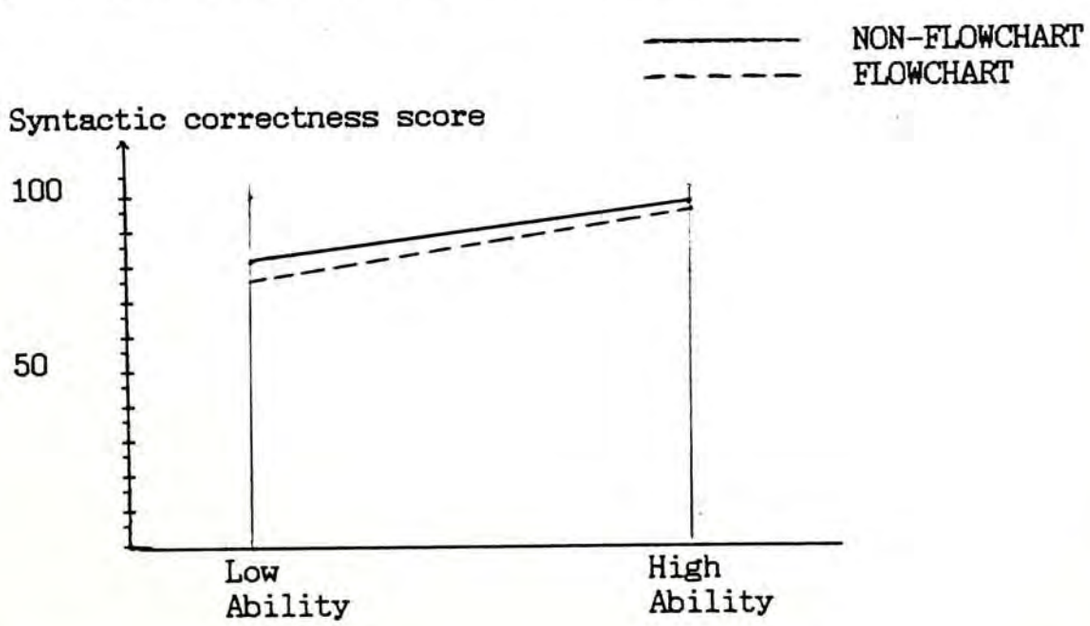


Figure 13 - Interaction of method of developing programs and ability of students (Syntactic correctness score of high-difficulty problem - Experiment II).

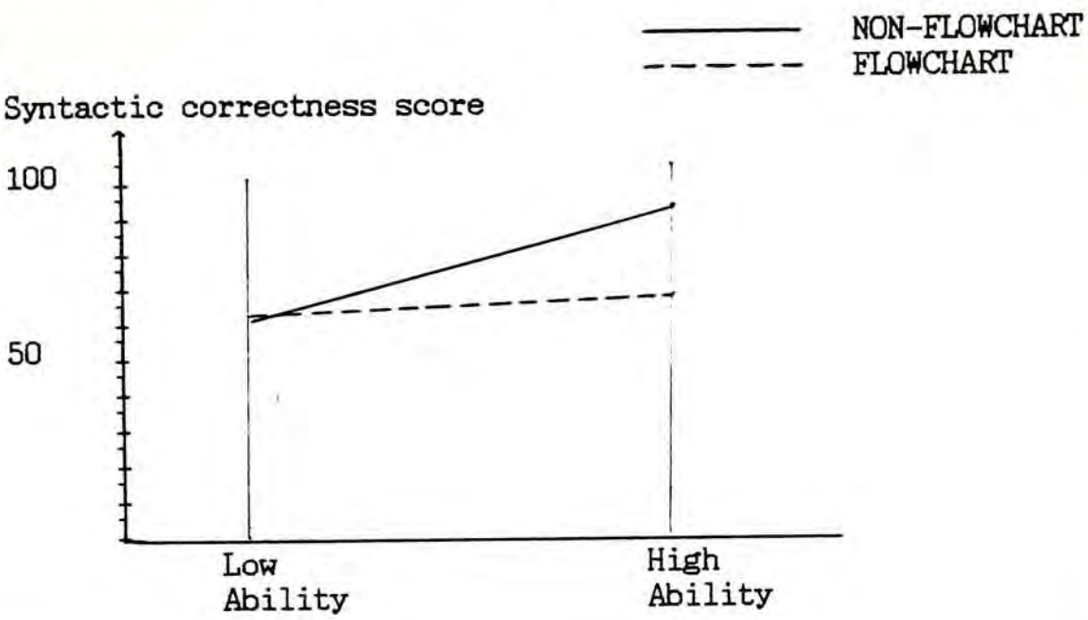


Figure 8 and Figure 13 showed that an interaction between method of developing programs and ability of students was observed, but not statistically significant. The results indicated flowcharting was more beneficial to the low-ability students while the high-ability students seemed to have a better programming performance when flowcharting was not required.

In the FLOWCHART group, the low-ability students were forced to organize the logic before writing lines of codes. This stage was essential since the low-ability students only had limited repertoire of templates, and they could not write down lines of codes directly by matching suitable

program templates. Thus the low-ability students should have a better programming performance when flowcharting is required.

On the other hand, the high-ability students seemed to have better programming performance when flowcharting was not required. The high-ability students had a large repertoire of program templates and they solved problem efficiently by active matching of suitable templates with subtasks. The stage of organizing program logic was bypassed for simple programming task. Flowcharting seemed to be a redundant job for the high-ability students. The high-ability students did not write flowcharts to organize program logic and they did not handle flowcharting well. In this experiment, the high-ability students were forced to use flowchart, they might find it difficulty to organize the program logic and the programming performance was lowered.

The sixth null hypothesis

As seen in Table 13, the male students performed generally better than the female students in all the three scores of both the low-difficulty and high-difficulty problems. Significant differences in F ratios were obtained for the semantic correctness scores of both the low-difficulty($F = 4.132$, $df = 1$, $p = .044$) and high-difficulty problems($F =$

4.605, df = 1, p = .033).

The results were in good accordance with those obtained in experiment one. Generally, male students had a more logical mind and showed a higher ability in writing programs.

As seen in table 14, the subjects with home computers performed generally better than the subjects without home computers. Significant differences were found in the logic scores($F(1,171) = 5.487$, $p < 0.05$) and the semantic correctness scores($F(1,171) = 7.528$, $p < 0.01$) of the low-difficulty problem.

Table 13 - An analysis of covariance of logic scores, semantic correctness scores and syntactic correctness scores by sex with pretest score as covariate

Score	Mean of Male (N = 112)	Mean of Female (N = 62)	df	F
LOGICA1	7.13(2.64)	5.47(2.95)	1/171	3.035 n.s.
LOGICA2	3.08(2.94)	2.03(4.05)	1/171	1.672 n.s.
SMNTA1	3.54(1.11)	2.81(1.27)	1/171	4.132 *
SMNTA2	1.77(1.37)	1.16(1.01)	1/171	4.605 *
SYNCORR1	92.79(13.03)	84.56(24.29)	1/171	1.054 n.s.
SYNCORR2	72.95(38.09)	63.16(42.00)	1/171	0.499 n.s.

* $p < 0.05$

n.s. - non-significant

Standard deviation enclosed in brackets.

Table 14 - An analysis of covariance logic scores, semantic correctness scores and syntactic correctness scores by HOMECPU with MARK as covariate

Logic Score	Mean of NOCPU group (N = 83)	Mean of CPU group (N = 91)	df	F
LOGICA1	5.98(3.03)	7.04(2.60)	1/171	5.487 *
LOGICA2	2.63(3.76)	2.78(3.05)	1/171	0.009 n.s.
SMNTA1	3.01(1.30)	3.53(1.09)	1/171	7.528 **
SMNTA2	1.48(1.14)	1.63(1.40)	1/171	0.213 n.s.
SYNCORR1	86.97(22.18)	92.49(13.28)	1/171	2.914 n.s.
SYNCORR2	67.98(39.17)	70.82(40.32)	1/171	0.052 n.s.

* $p < 0.05$; ** $p < 0.01$
n.s. - non-significant
Standard deviation enclosed in brackets.

Students possessing home computer certainly had more chance to use the computers. The students might not use the computer for programming activities, but, when they operated the computers, they had to use the language of the operating system. They should know the semantics and syntax of the command lines when they keyed into the computer. In order to complete a certain task, they had to key in several command lines in a logical order. This was the concept of programming. As a result, these students had more training in writing programs. On the other hand, students possessing home computers were more familiar with computers, thus, they had a low degree of anxiety towards computer. A low computer anxiety might cause a better programming performance(Loyd

& Gressard, 1984),therefore, students with home computers should have better performance in programming.

4.2.4 The attitude towards use of flowcharts in developing programs

The last four questions(questions 8 to 11) in the questionnaire served to collect the information about the attitude of the subjects towards use of flowcharts in the process of developing a program. The subjects responded to these four question in a five-point Likert scale. The four questions were shown in Appendix 11. Question 10 and 11 were recoded in the reverse direction.

As seen in Table 15, the mean score for question 8 was relatively low and this indicated that most of the students seldom draw flowcharts to organize the logic before writing programs. Similar results were found in other researches(Dalbey, Tourniaire, & Linn, 1986; McCormick & Ross, 1990; Shneiderman, Mayer, McKay & Heller, 1977).

A score showing the attitude of the subject towards use of flowcharts in the process of developing a program (FLOWATTD) was calculated by summing up the scores of the four questions. Thus FLOWATTD ranged from 4 to 20. The mean and standard deviation of FLOWATTD were 11.2292(N =

336) and 3.2237, respectively. The result was slightly lower than the medium score(12.0) and this indicated that the students had a slightly negative attitude towards the use of flowcharts in the development of programs.

Table 15 - The results of questions concerning the attitude towards use of flowcharts in developing programs

Question number	Mean (N = 336)	Standard deviation
8	1.8274	0.9011
9	3.1161	1.1951
10	3.1429	1.2157
11	2.9792	1.3723

Table 16 - Correlation results of different scores with FLOWATTD

	Variables	FLOWATTD
Experiment 1	Pretest score	-.2191 *
	LOGICA1	-.1829 *
	LOGICA2	-.1985 *
Experiment 2	Pretest score	-.1397
	LOGICA1	-.0435
	LOGICA2	-.1397
	SMNTA1	-.0292
	SMNTA2	-.1114
	SYNCORR1	-.0858
	SYNCORR2	-.2045 *

* $p < 0.05$

Interesting results were found when the different scores were correlated with FLOWATTD(Table 16). The results showed that FLOWATTD was negatively correlated to all of the dependent variables, but most of them were not statistically significant. This implied that students who preferred the use of flowcharts in developing programs might have a lower programming performance.

High-ability students wrote program lines directly by matching suitable templates in the large repertoire of program templates stored in their long term memory. They might not appreciate the functions of flowcharts, thus, they might have a negative attitude towards the use of flowcharts in developing programs.

For the low-ability students, they could not write down program lines directly owing to a limited repertoire of program templates. They had to figure out the program logic before writing lines of codes. The low-ability students might think that flowcharting might help them in the organization of program logic, hence, they might have a more positive attitude towards use of flowcharts in developing programs.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Summary of findings

5.1.1 Instruments

The reliability coefficients of the pretest were found quite high . The interrater correlation coefficients were extremely high. Thus, the instruments did have high internal consistency.

5.1.2 Experiment I

Significant differences in logic scores were found between subjects of low and high ability, and between subjects of using different tools to construct program logic. High-ability subjects performed better than the low-ability subjects, and subjects using BASIC language performed better than the subjects using flowcharts in the construction of program logic.

No significant interaction was found between program organization tools and ability of students both in the low-

difficulty and high-difficulty problems.

Slight gender difference was found in the construction of program logic. Male students were found to have high logic scores, with significance shown in the low-difficulty problem at .05 confidence level.

Students with home computer performed better in the low-difficulty problem, but about the same in the high-difficulty problem. No significant difference was found in the logic scores between the two groups of students.

5.1.3 Experiment II

Significant differences in logic scores were found between subjects of low and high ability, and between subjects of using different tools to construct program logic. High-ability subjects performed significantly better in all the three scores than the low-ability subjects. Subjects of the NON-FLOWCHART group performed better generally than subjects of the FLOWCHART group, with significant difference found in the logic score and semantic correctness score of the high-difficulty program.

No significant interaction was found between the FLOWCHART group and NON-FLOWCHART group between method of developing programs and ability of students in all the

scores both of the low-difficulty and high-difficulty problems.

Male students were found to have higher scores than the female students in programming performance, with significant difference shown in the semantic correctness scores of both the low-difficulty and high-difficulty problems at .05 confidence level.

Students with home computer generally performed better in the low-difficulty problem, but about the same in the high-difficulty problem. Significant difference was found in the logic score and semantic correctness score of the low-difficulty between the two groups of students.

5.2 Conclusions

The purpose of this study is to examine the effect of flow-charting on the programming composition skills of students. Of all the null hypotheses set, the data collected managed to reject only a few of them. Significant difference was found on both the logic scores of the low-difficulty and high-difficulty problems between subjects in the BASIC group and the FLOWCHART group. The results suggested that students constructed program logic better when the programming language itself was used. Flowchart seemed

to have a non-significant effect on the construction of program logic. This findings was in good accordance with the study done by McCormick and Ross(1990).

In the examination of the results of the attitude of subjects towards flowcharting, it is found that most of the subjects did not appreciate the functions of the flowchart in the development of programs(Table 15). The subjects tended to construct the program first and then write the flowchart accordingly when flowcharts are required to submit together with the program.

For the second experiment, although it was found that there was significant difference in the logic score the high-difficulty problem between the FLOWCHART group and the NON-FLOWCHART group, the difference was not so obvious as that shown in Experiment I. During the experiment, the subjects in the FLOWCHART group were asked to draw a flowchart first and then wrote the programming accordingly. This was only the ideal working procedure for the FLOWCHART group. In some of the raw script, the BASIC program was written on the first page, and the flowcharts was drawn on the second page. It suggested that, these subjects wrote the program first and then drew the flowchart separately or accordingly, as revealed in the attitude towards use of flowcharts in the development of programs(Table 15). Some of the subjects might draw the flowchart first, but wrote the program separately, ignoring the logic developed by flowcharting. Therefore, the effect of flowcharting was not shown completely in the programming performance,

hence only significant difference was found in the logic score of the high-difficulty problem between the two groups.

No significant interaction effect was found in both experiments. Thus, effect of flowcharting seems to have similar effect for both the high-ability and low-ability students.

In this research, the ability of the subjects was most strongly and consistently related to programming performance which is measured in terms of the logic score, semantic correctness score and the syntactic correctness score. Gender differences among subjects were related to some aspects of programming performance, logic and semantics. It was found that male students generally performed better than female students in program composition. Home computer possession seemed to reduce the computer anxiety and leads to a better performance in both experiments.

5.3 Limitations

During the process of the experiment, a few problems were encountered and these problems might affect the internal and external validity of this research study. These limitations also implied that interpretations of the research findings must be taken with great care.

5.3.1 The sampling problem

The subjects in this research were not randomly sampled. They were selected at the convenience of the researcher. As a compromise between limitation of resources and sample size, only 365 subjects were selected in this researcher. The limited size of the sample might not be large enough to have a good external validity. Therefore, the generalization of the results of this research was limited.

5.3.2 The grouping problem

The subjects in this research were matched in pairs and divided into two groups of equal ability according to the pretest scores. A few number of the subjects were absent from the experiment after taking the pretest. These subjects were deleted from the sample pool and this made the numbers of subjects unequal between the two groups in the experiments. This factor might lower the internal validity of the research.

5.3.3 The instrument development

The instruments for measuring the knowledge in BASIC

language and flowcharts, program logic, semantic correctness and syntactic correctness were newly developed by the researcher for this study. Since each of the tests had to be completed within 70 minutes(a double lesson), the problems were designed to meet the constraint of time. The level of difficulties of the problems were carefully set to avoid a ceiling effect to the results. A pilot study had been conducted and refinements had been made to each of the instruments. Though the reliability of the instruments was improved, further refinements were still needed.

5.4 Recommendations

Results from this research indicate that flowcharts may not be the most suitable tool for the development of programs. Previous results showed that a program organization phase was essential for good programming performance. Students were encouraged to organize the program logic before writing the program. Most students tend to have their organized plan in mind, but it does not seem helpful in the development of programs. Some kinds of tools should be used to figure out the logic and prompt a completion of logic. Flowcharts seem to have outweighed by their short-comings. Detailed flowcharts are merely a redundant presentation of the information stored in programs. Programming languages are more concise in presenting the semantics and logic.

Utilization of top-down design techniques may reduce the use of detailed flowcharts. A large repertoire of program templates facilitates flowcharting in macro level such that chunks of codes can be represented by functional boxes and major flow of control is indicated by flow lines. Macro flowcharts was found to alleviate anxiety and confusion when subjects were given large programs(Shneiderman, Mayer, McKay, & Heller, 1977). Thus, students with macro flowcharts may have a better performance in comprehension for large programs. In program composition, the size of chunks encoded in the functional boxes depend much on the ability of students. Macro flowcharts can not be encoded into program lines if no suitable template is found to match with the functional box. Thus, macro flowcharts seem to have different effect on expert and novice programmers. Further work is necessary to explore the use of macro flowcharts in developing programs.

In summary, the experiments have demonstrated that flowcharting shows no significant effect on program composition. It is suggested that a better program organization tool should be searched, or a number of different organization tools should be introduced to the students such that the students can find an organization tool that suits them most. Further research should be carried out to investigate the effects of different program organization tools.

Bibliography

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. Memory and Cognition, 9, 422-433.

Bayman, P., & Mayer, R.E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. Communications of the ACM, 26(9), 677-679.

Boulay B. D. (1986). Some difficulties of learning to program. Journal of Educational Computing Research, 2(1), 57-73.

Brooks, R.E. (1980). Studying programmer behavior experimentally: the problems of proper methodology. Communications of the ACM, 23(4), 207-214.

Carey, J.M., & McLeod, R. (March, 1988). Use of system development methodology and tools. Journal of Systems Management, 30-36.

Collins, R.W., & White, K.B. (Spring, 1980). An investigation of componential skill relationships in programming. AEDS Journal, 123-129.

Curriculum Development Committee, Hong Kong. (1986). Syllabuses for secondary schools - Computer Studies (Forms IV - V), 7-8.

Dalbey, J., Tourniaire, F., & Linn, M.C. (1986). Making programming instruction cognitively demanding : an intervention study. Journal of Research in Science Teaching, 23, 421-436.

Gasen, J.B., & Morecroft, J.F. (1990). Hemispheric lateralization and programming ability. Journal of Educational Computing Research, 6(1), 17-28.

Hall, J., & Cooper, J. (1991). Gender, experience and attributions to the computer. Journal of Educational Computing Research, 7(1), 51-60.

Keeler, J.K. (1990). Characteristics of LOGO instruction promoting transfer of learning: a research review. Journal of Research on Computing in Education, 23(1), 55-71.

Kolb, D.A., & Raben, I.M. (1971). Organizational psychology. New York. Prewtice Hall. 34-44.

Kurland, D.M., Pea, R.D., Cleement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. Journal of Educational Computing Research, 2(4), 429-458.

Linn, M.C., & Dalbey, J. (1985). Cognitive consequences of programming instruction : instruction, access, and ability. Educational Psychologist, 20(4), 191-206.

Loyd, K.H., & Gressard, C. (Winter, 1984). The effects of sex, age, and computer Experience on Computer Attitudes. AEDS Journal, 67-77.

Madeo, L.A., & Bird, D.A. (1990). The effect of user-specified language elements on learning to program. Journal of Research on Computing in Education, 23, 337-347.

Marcoulides, G.A. (1988). The relationship between computer anxiety and computer achievement. Journal of Educational Computing Research, 4(2), 151-157.

Mayer, R.E. (1979). A psychology of learning BASIC. Communications of the ACM, 22(11), 589-593.

Mayer, R.E. (1981). The psychology of how novices learn computer programming. Computing Surveys, 13(1), 121-141.

McAllister, D.W., & Brock, F.J., Jr. (1990). Using the logic diagram in teaching programming : an overview and preliminary test results. Journal of Research on Computing in Education, 22(3), 348-363.

McCormick, D., & Ross, S.M. (1990). Effects of computer access and flowcharting on students' attitudes and performance in learning computer programming. Journal of Educational Computing Research, 6(2), 203-214.

McCoy, L.P. (1991). Computer programming experience and mathematical problem solving. Journal of Research on Computing in Education, 6(1) 14-24.

Pea, R.D. (1986). Language-independent conceptual "bugs" in novice programming. Journal of Educational Computing Research, 2(1), 25-36.

Palumbo, D.B., & Reed, W.M. (1991). The effect of BASIC programming language instruction on high school students' problem solving ability and computer anxiety. Journal of Research on Computing in Education, 23(3), 343-372.

Perkins, D.N., Hancock, C., Hobbs, R., Martin, R., & Simmons, R. (1986). Conditions of learning in novice programmers. Journal of Educational Computing Research, 2(1), 37-55.

Pohl, H.L., & Nutter, J.T. (Summer, 1985). Use of analogy in computer language acquisition. AEDS Journal, 254-266.

Ramsey, H.R., Atwood, M.E., & Van Doren, JCR. (1983). Flowcharts versus program design languages : an experimental comparison. Communications of the ACM, 26(6), 445-449.

Salomon, G., & Perkins, D.N. (1987). Transfer of cognitive skills from programming : when and how? Journal of Educational Computing Research, 3(2), 149-169.

Sectional Committee X3, Computers and Information Processing, operating under the auspices and procedure of the American Standards Association (1963). Proposed American standard flowchart symbols for information processing. Communications of the ACM, 6(10), 601-604.

Shaw, D.G. (Winter/Spring, 1986). Effects of learning to program a computer in BASIC or LOGO on problem-solving abilities. AEDS Journal, 176-189.

Shneiderman, B. , Mayer, R., McKay, D., & Heller, B. (1977). Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM, 20(6), 373-381.

Shute, V.J. (1991). Who is likely to acquire programming skills? Journal of Educational Computing Research, 7(1), 1-24.

Stemer, L. (1989). Effects of instruction on the misconceptions about programming in BASIC. Journal of Research on Computing in Education, 22(1), 26-34.

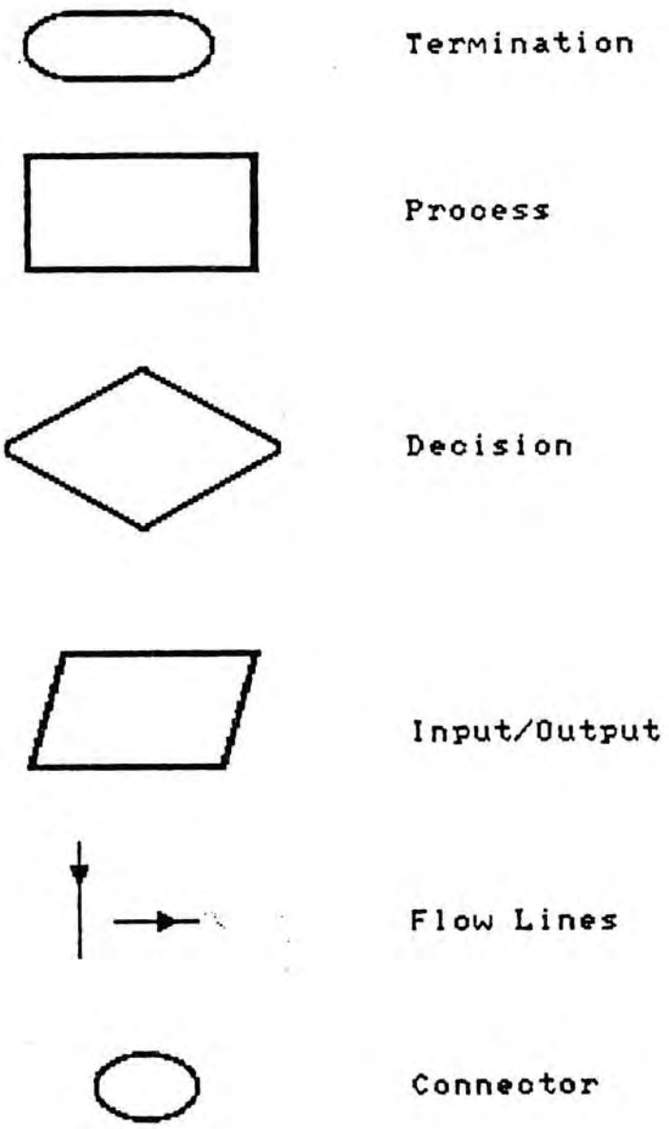
Van Merrienboer, Jeroen J.G. (1990). Instructional strategies for teaching computer programming : interations with the cognitive style reflection impulsivity. Journal of Research on Computing in Education, 23(1), 34-54.

Van Merrienboer, Jeroen J.G. (1990). Strategies for programming instruction in high school : program completion vs program generation. Journal of Educational Computing Research, 6(3), 265-286.

Yu Y. T. (1989). Elements of BASIC programming - a problem solving approach. Hong Kong : Modern Educational Research Society Ltd., pp61-62.

Appendix 1

Set of Flowchart Symbols



Appendix 2

Requirements of a good program design tool(McAllister and Brock, 1990)

A good program design tool for classroom use needs to :

1. communicate program hierarchical organization, modularity, and detail;

2. attract attention visually through its graphical simplicity (straight lines, few angles and symbols);

3. show the three types of control structure : sequence, decision, and looping;

4. trace the dynamics of the program by flowing from top to bottom, and by showing the change of control from one module to another;

5. prompt complete logic;

6. facilitate documentation, by hand, word processor, or spreadsheet; and,

7. encourage further abbreviation and integration with other techniques.

Appendix 3

Level of knowledge in programming (Mayer, 1979)

1. Machine :

   Each specific, single change that may occur within the actual hardware of the computer.

2. Transaction :

   Related to the general functions of the computer and is the building block from which statements are made.

3. Prestatement :

   The subcategories of statements.

4. Statement :

   A class of one or more prestatements all sharing the same name.

5. Mandatory chunks :

   A series of two or more statements that must occur in some configuration. e.g. FOR / NEXT.

6. Basic nonmandatory chunks :

   A series or configuration of prestatements that is often used in a variety of programs to accomplish some general goal.

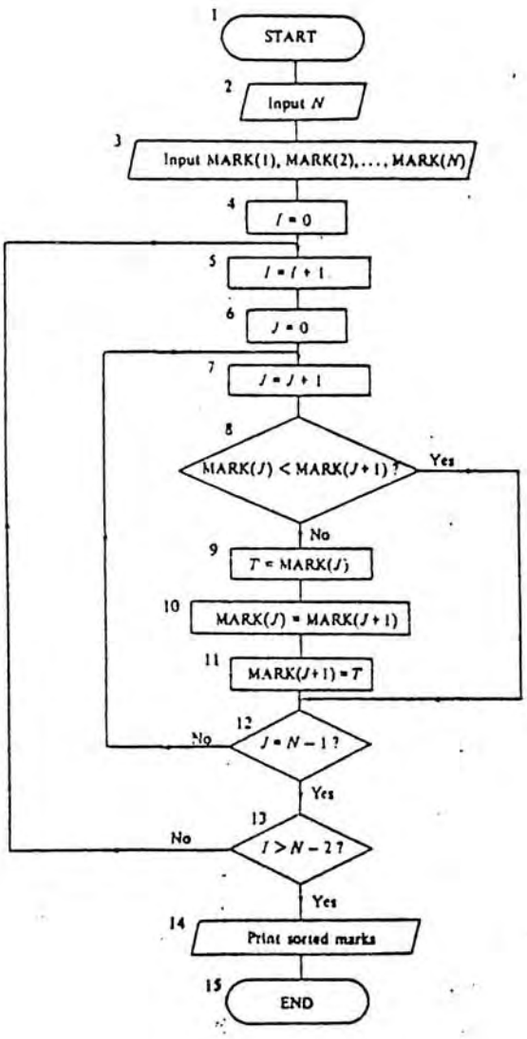
7. Higher nonmandatory chunks :

   an extension of basic nonmandatory chunks.

8. Programs :

   The highest level of knowledge, can be directly analyzable into a set of chunks and statements.

Appendix 4
Example of Flowchart (Computer Studies II, HKCEE 1986)



Appendix 5
Example of program (Computer Studies II, HKCEE 1987)

1. The Computer Club of a school assigns identity codes to its members in the following manner :

   the first character is a number representing the form in which the student is studying;

   the second character is a letter representing the class in the form in which the student is studying;

   the third and fourth characters represent the student's two-digit class number;

   the fifth character is a check digit (see figure 2).

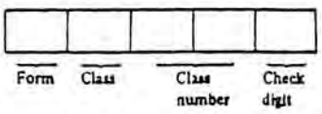


Figure 2

When a student wants to take part in Computer Club activities, his identity code has to be checked. The following program is written to check the validity of an inputted code.

(Assume that the program can be run in the machine used.)

```
100 REM *** PROGRAM TO CHECK MEMBERSHIP ***
110 DIM E$(6)
120 FOR I=1 TO 5
130     READ E$(I)
140 NEXT I
150 D=1
160 REM *** GET STUDENT ID CODE ***
170 INPUT "WHAT IS THE CODE TO BE CHECKED";C$
180 IF LEN(C$)<>5 THEN 170
190 REM *** F=FORM ***
200 REM *** G$=CLASS NAME ***
210 REM *** S=STUDENT'S CLASS NUMBER ***
220 REM *** C=CHECK DIGIT ***
230 REM *** EXTRACT INFORMATION FROM INPUT CODE ***
240 F=VAL(LEFT$(C$,1))
250 G$=MID$(C$,2,1)
260 S=VAL(MID$(C$,3,2))
270 C=VAL(RIGHT$(C$,1))
280 REM *** CHECK THE EXTRACTED INFORMATION ***
290 IF F<=7 AND F>=1 THEN 320
300 D=2
310 GOTO 410
320 IF G$<="D" AND G$>="A" THEN 350
330 D=3
340 GOTO 410
350 IF S<=50 AND S>=1 THEN 380
360 D=4
370 GOTO 410
380 A=VAL(MID$(C$,3,1))+VAL(MID$(C$,4,1))+VAL(MID$(C$,5,1))
390 IF A=INT(A/10)*10 THEN 410
400 D=5
410 REM *** PRINT MESSAGE ***
420 PRINT
430 IF D=1 THEN PRINT"STUDENT";S;"OF ";LEFT$(C$,2);" ";
440 PRINT E$(D)
450 END
460 REM *** DATA STATEMENTS ***
470 DATA IS A MEMBER
480 DATA NO SUCH FORM
490 DATA NO SUCH CLASS
500 DATA NO SUCH CLASS NUMBER
510 DATA THE STUDENT IS NOT A MEMBER
```

Appendix 6

Questions of the Pretest

Ref no. ___

TEST - FLOWCHARTING AND BASIC PROGRAMMING LANGUAGE

NAME : _____(English - in block letters)

CLASS : _____ Class No. : _____

Answer ALL questions
Time allowed is 45 minutes

Write the answer in the following boxes.

	1	2	3	4	5	6	7	8	9	10
answer :										

	11	12	13	14	15	16	17	18	19	20
answer :										

	21	22	23	24	25	26	27	27	29	30
answer :										

A partial Character List for ASCII

Character	ASCII	Character	ASCII	Character	ASCII
0	48	I	73	b	98
1	49	J	74	c	99
2	50	K	75	d	100
3	51	L	76	e	101
4	52	M	77	f	102
5	53	N	78	g	103
6	54	O	79	h	104
7	55	P	80	i	105
8	56	Q	81	j	106
9	57	R	82	k	107
:	58	S	83	l	108
;	59	T	84	m	109
<	60	U	85	n	110
=	61	V	86	o	111
>	62	W	87	p	112
?	63	X	88	q	113
@	64	Y	89	r	114
A	65	Z	90	s	115
B	66	[	91	t	116
C	67	\	92	u	117
D	68	]	93	v	118
E	69	^	94	w	119
F	70	_	95	x	120
G	71	`	96	y	121
H	72	a	97	z	122

Ref no. ___

TEST - FLOWCHARTING AND BASIC PROGRAMMING LANGUAGE

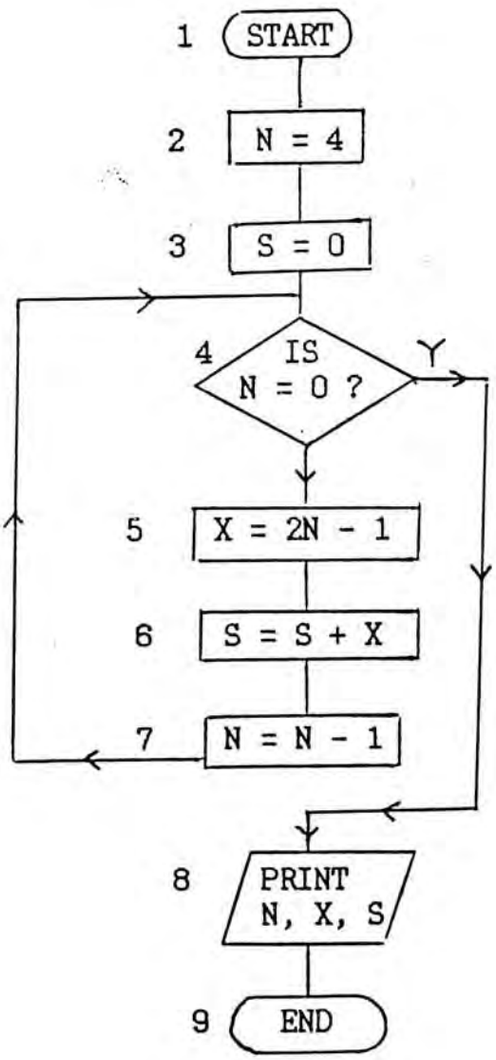
NAME : _____(_____)

CLASS : _____ Class No. : ____

Answer ALL questions

Time allowed is 45 minutes

Questions 1 to 4 refer to the flowchart below.



1. What is the value of X at box 5 when the control reaches box 5 at the SECOND time?

A. 1 B. 3 C. 5
D. 7 E. 9

2. What is the value of S when the control reaches box 8?

A. 2 B. 4 C. 8
D. 16 E. 32

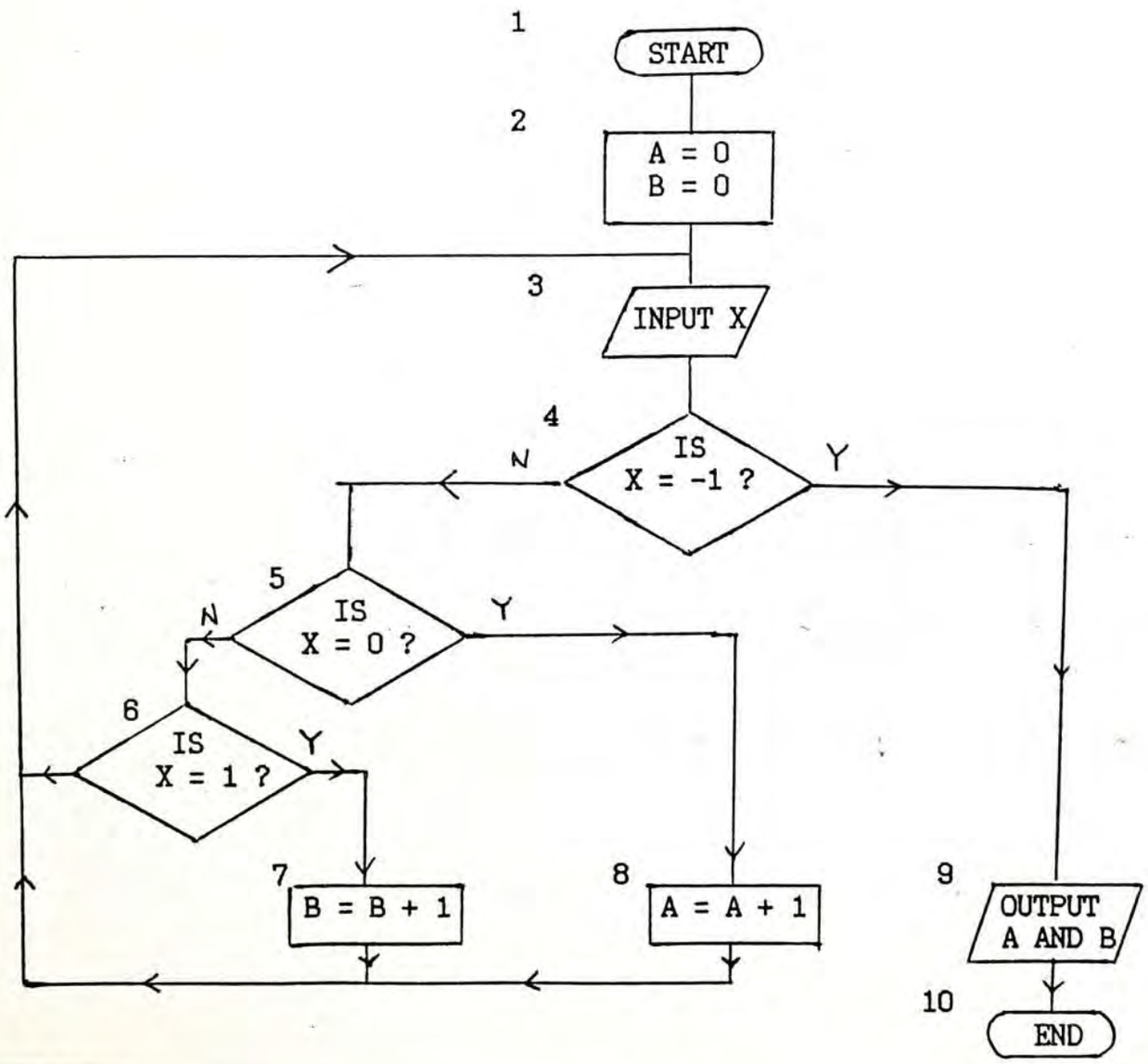
3. What is the value of N when the control reaches box 8?

A. 0 B. 1 C. 2
D. 3 E. 4

4. What is the value of S at box 8 if the content of box 2 is changed to N=5 ?

A. 16 B. 25 C. 32
D. 36 E. 45

Questions 5 to 8 refer to the flowchart on next page. The data to be inputted are 0, 1, 2, 0, 1, 1, 2, 0, -1 and -2.



5. What is the value of A when the control reaches box 9?

A. -1 B. 0 C. 1
D. 2 E. 3

6. What is the value of B when the control reaches box 9?

A. -1 B. 0 C. 1
D. 2 E. 3

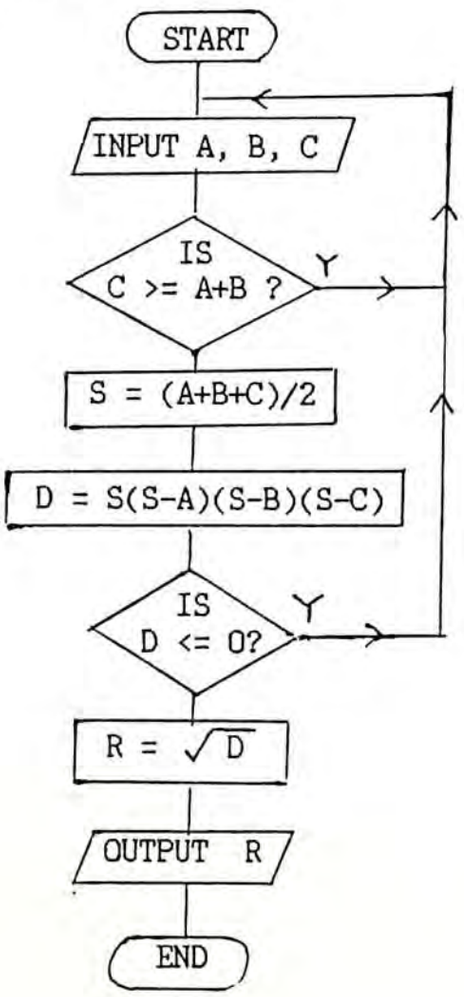
7. What is the value of X when the control reaches box 9?

A. -1 B. 0 C. 1
D. 2 E. 3

8. What is the objective of the flowchart ?

A. to determine whether the data inputted is zero, positive or negative.
B. to count the number of '1's.
C. to count the number of '1's and the number of '0's.
D. to count the number of '1's , '0's and '-1's.
E. to count the total number of '1's and '0's.

9. Consider the followng flowchart.



Which of the following program is the correct coding for the above flowchart ?

A.
```
100   INPUT A, B, C
110   IF C >= A+B THEN GOTO 100
120   S=(A+B+C)/2
130   D = S*(S-A)*(S-B)*(S-C)
140   IF D <= 0 THEN GOTO 110
150   R = SQR(D)
160   PRINT R
170   END
```

B.
```
100   INPUT A, B, C
110   IF C >= A+B THEN GOTO 170
120   S=(A+B+C)/2
130   D = S*(S-A)*(S-B)*(S-C)
140   IF D <= 0 THEN GOTO 110
150   R = SQR(D)
160   PRINT R
170   END
```

C.
```
100   INPUT A, B, C
110   IF C >= A+B THEN GOTO 170
120   S=(A+B+C)/2
130   D = S*(S-A)*(S-B)*(S-C)
140   IF D <= 0 THEN GOTO 170
150   R = SQR(D)
160   PRINT R
170   END
```

D.
```
100   INPUT A, B, C
110   IF C >= A+B THEN GOTO 170
120   S = S*(S-A)*(S-B)*(S-C)
130   S = (A+B+C)/2
140   IF D <= 0 THEN GOTO 110
150   R = SQR(D)
160   PRINT R
170   END
```

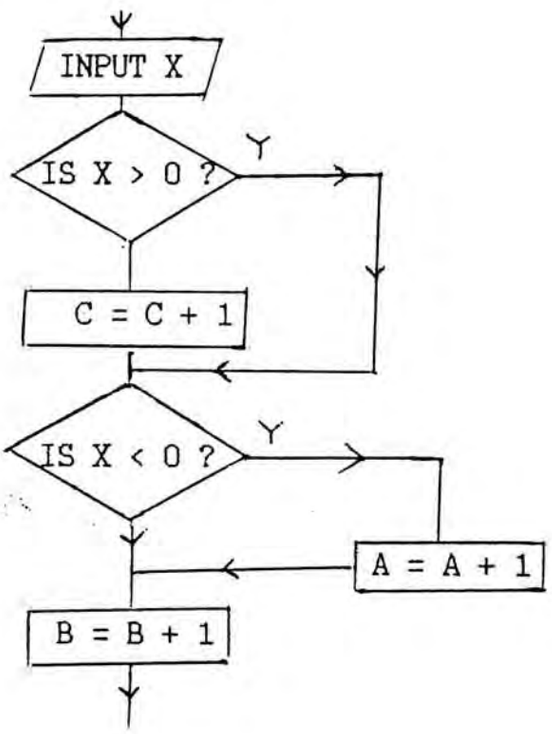
E.
```
100   INPUT A, B, C
110   IF C >= A+B THEN GOTO 100
120   S=(A+B+C)/2
130   D = S*(S-A)*(S-B)*(S-C)
140   IF D <= 0 THEN GOTO 100
150   R = SQR(D)
160   PRINT R
170   END
```

10. Consider the following program segment.

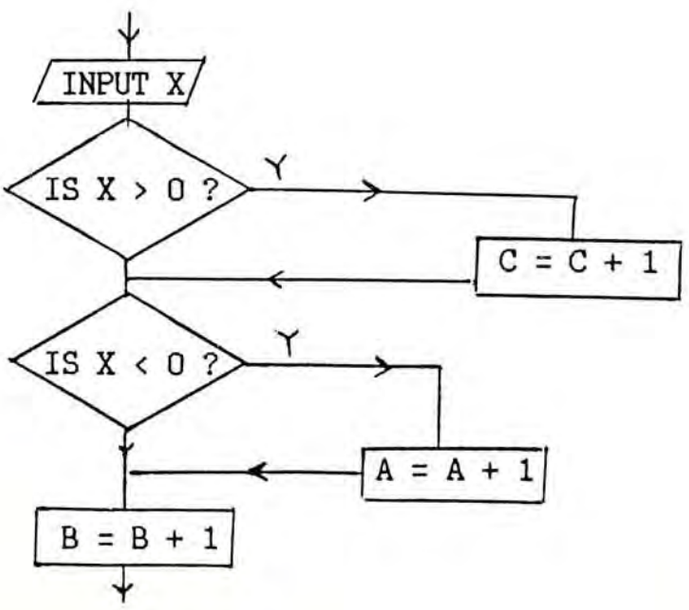
```
:
200   INPUT X
210   IF X > 0 THEN C = C + 1
220   IF X < 0 THEN B = B + 1 ELSE A = A + 1
230 ...
:
:
```

Which of the following flowchart structures represents the same logic with the above program segment?

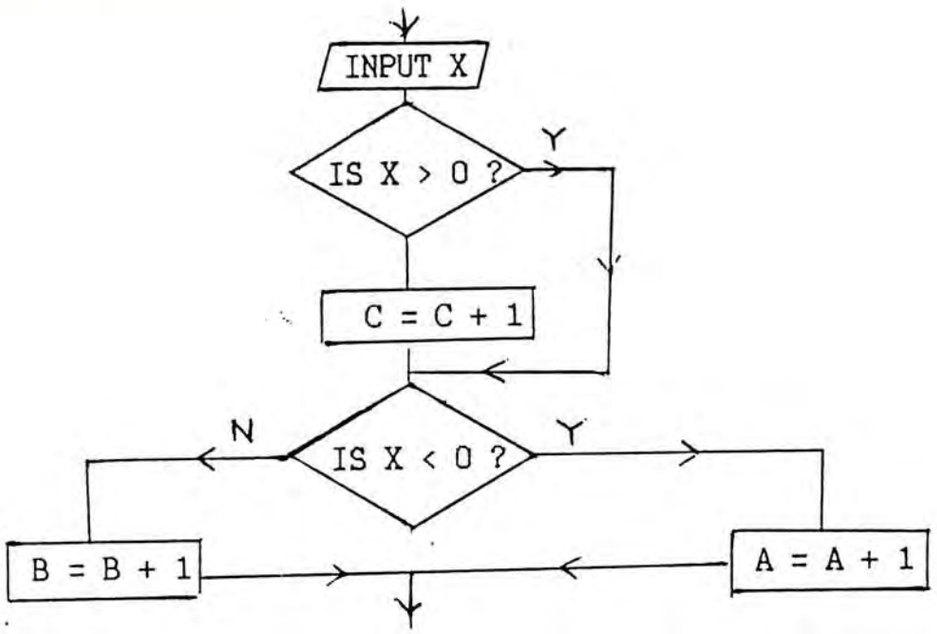
A.



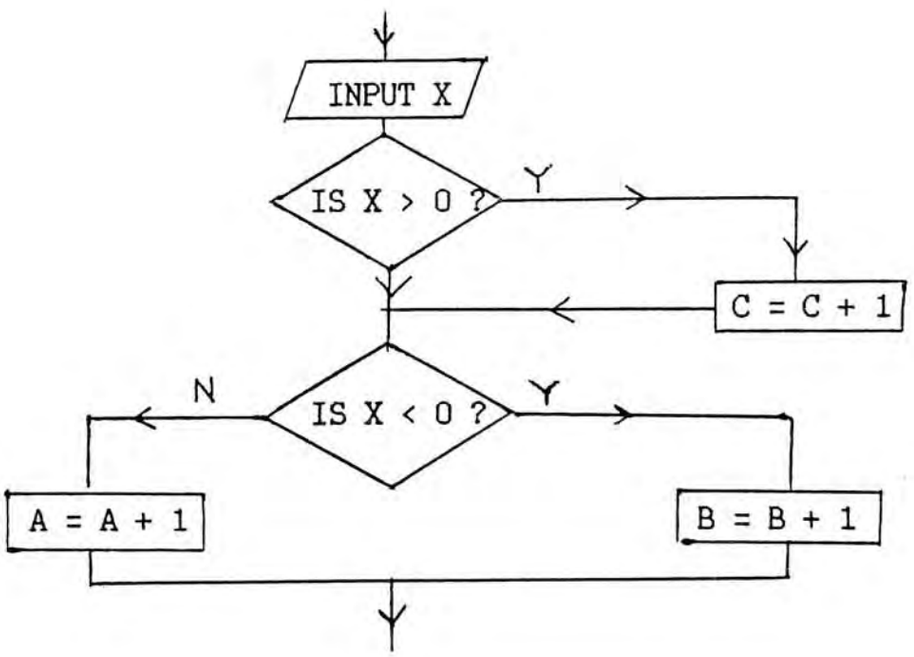
B.



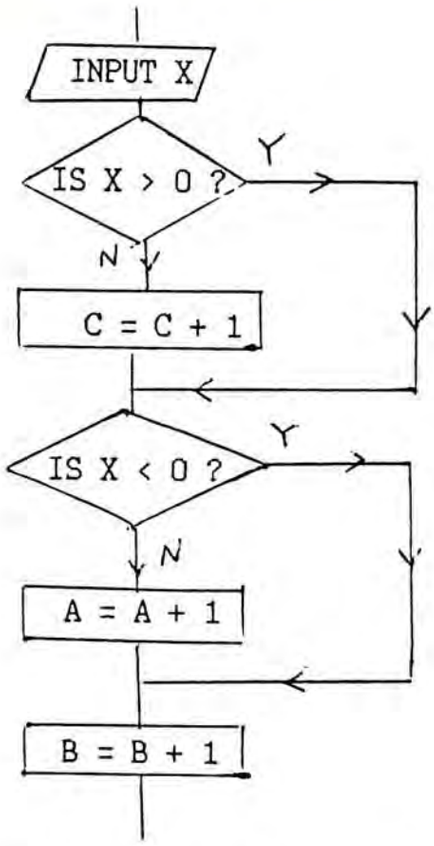
C.


INPUT X
IS X > 0 ?
Y
C = C + 1
N
IS X < 0 ?
Y
B = B + 1
A = A + 1

D.


INPUT X
IS X > 0 ?
Y
C = C + 1
N
IS X < 0 ?
Y
A = A + 1
B = B + 1

E.



11. Which of the following BASIC statements is free of syntax error?

A. 10 C = 2C
B. 10 C > 5 THEN 100
C. 10 REM STOP : STOP
D. 10 C + 1 = C
E. 10 C = SIN X

12. Consider the following equation.

$$Y = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

Which of the following statements is the correct coding for the above equation ?

A. Y = -B + SQR(B*B) - 4*A*C / 2 * A
B. Y = -B + (SQR(B*B) - 4*A*C) / (2 * A)
C. Y = (-B + SQR(B*B) - 4*A*C) / (2 * A)
D. Y = (-B + SQR(B*B - 4*A*C)) / (2 * A)
E. Y = (-B + (SQR(B*B) - 4*A*C)) / (2 * A)

13. Consider the following program. What is the value stored in A(3,2)?

```
10    DIM A(3,3)
20    FOR I = 1 TO 3
30    FOR J = 1 TO 3
40    READ A(I,J)
```

```
50    NEXT J
60    NEXT I
70    DATA 1,2,3,4,5,6,7,8,9
80    END
```

A. 2 B. 4 C. 5
D. 6 E. 8

14. Consider the following program. How many times will the word "TESTING" be printed when the program is executed?

```
10    FOR I = 1 TO 6
20    FOR J = 1 TO 6 STEP 2
30    PRINT "TESTING"
40    NEXT J
50    NEXT I
60    END
```

A. 3 B. 6 C. 9
D. 18 E. 36

15. Consider the following program. How many times will the word "TESTING" be printed when the program is executed?

```
10    FOR H = 10 TO 1 STEP -2
20    FOR K = 1 TO 3
30    PRINT "TESTING"
40    NEXT K
50    NEXT H
60    END
```

A. 15 B. 18 C. 24
D. 30 E. 60

16. Consider the following program. What is the value stored in SUM(2) after the execution of the program?

```
10    DIM SUM(3)
20    FOR H = 1 TO 3
30    SUM(H) = 0
40    NEXT H
50    FOR J = 1 TO 6
60    READ N, AMT
70    SUM(N) = SUM(N) + AMT
80    NEXT J
90    DATA 1, 10, 1, 20, 3, 15, 2, 18, 2, 32, 3, 45
100   END
```

A. 30 B. 25 C. 50
D. 33 E. None of the above

17. The assignment statement to round off a real number X to 2 decimal places should be

A. `Y = INT(X + 0.5) + 0.05`
B. `Y = INT(X + 0.5) + 0.01`
C. `Y = INT(X * 100)*0.01 + 0.5`
D. `Y = INT(X * 100 + 0.5) * 0.01`
E. `Y = (INT(X + 0.5) * 100 + 0.5) * 0.01`

18. Which of the following statements will return a "*" if X is <u>even</u>?

A.
```
10    IF INT(X) = X THEN PRINT "*"
```
B.
```
10    IF INT(X/2) = X/2 THEN PRINT "*"
```
C.
```
10    IF X/2 = INT(X) THEN PRINT "*"
```
D.
```
10    IF INT(X) * 2 = X * 2 THEN PRINT "*"
```
E.
```
10    IF INT(X)/ 2 = X / 2 THEN PRINT "*"
```

19. Consider the following program. What will be the value of X if the program is executed?

```
10    READ A, B
20    C = INT(A/B)
30    X = A - C * B
40    PRINT X
50    DATA 35, 6
60    END
```

A. 1 B. 3 C. 5
D. 7 E. None of the above

20. Consider the following program. What is the output when the program is executed?

```
10    READ A, B
20    RESTORE
30    READ C, D
40    PRINT A, B, C, D
50    DATA  10, 20, 30, 40
60    END
```

A. 10, 20, 30, 40 B. 10, 10, 20, 20
C. 10, 10, 40, 40 D. 10, 20, 10, 20
E. 20, 10, 20, 10

21. Consider the following program. What is the output when the program is executed?

```
10    READ A, B
20    C = A + B * B
30    C = C - A
40    PRINT C
50    DATA  2, 4
60    END
```

A. 2 B. 8 C. 14
D. 16 E. 18

22. Consider the following program segments. Which of the program segments will swap the values of A and B (i.e. exchange the values stored in A and in B) ?

PROGRAM SEGMENT I

```
60 A = B
70 B = A
```

PROGRAM SEGMENT II

```
60 C = A
70 A = B
80 B = C
```

PROGRAM SEGMENT III

```
60 C = A
70 D = B
80 A = D
90 B = C
```

A. I only B. III only C. I and II only
D. II and III only E. I, II and III

23. Consider the following program. What is the output when the program is executed?

```
10    A$ = "HELLO"
20    B$ = "MARY"
30    A$ = A$ + "!"
40    B$ = "  " + B$
50    PRINT A$ + B$
60    END
```

A. A$ + B$ B. MARY HELLO!
C. HELLO! MARY D. HELLO + ! + MARY
E. None of the above

24. Consider the following program.

```
10    A$ = "COMPUTER"
20    _____________
30    PRINT "THE NEW WORD IS ";B$
40    END
```

Which of the following statement should be put into line 20 in order to produce the following result?

THE NEW WORD IS PUT

A. B$ = A$ - "COMER" B. LEFT$(RIGHT$(A$,5),3)
C. B$ = MID$(A$,3) D. B$ = MID$(A$, 3, 4)
E. B$ = MID$(A$, 4, 3)

25.

```
A
AP
APP
APPL
APPLE
```

Which of the following program will produce the above pattern of output?

A.
```
10    READ X$
20    FOR H = 1 TO VAL(X$)
30    PRINT RIGHT$(X$,H)
40    NEXT
50    DATA "APPLE"
60    END
```

B.
```
10    READ X$
20    FOR H = 1 TO VAL(X$)
30    PRINT LEFT$(X$,H)
40    NEXT
50    DATA "APPLE"
60    END
```

C.
```
10    READ X$
20    FOR H = 1 TO VAL(X$)
30    PRINT MID$(X$, H , H)
40    NEXT
50    DATA "APPLE"
60    END
```

D.
```
10    READ X$
20    FOR H = 1 TO LEN(X$)
30    PRINT RIGHT$(X$, H)
40    NEXT
50    DATA "APPLE"
60    END
```

E.
```
10    READ X$
20    FOR H = 1 TO LEN(X$)
30    PRINT LEFT$(X$, H)
40    NEXT
50    DATA "APPLE"
60    END
```

26. Consider the following program. What is the output when the program is executed?

```
10    READ A, B
20    IF A > 0 OR B > 0 THEN PRINT A ELSE PRINT B
30    DATA 3, 0
40    END
```

A. 3
B. 0
C. 3 0
D. 0 3
E. None of the above

27. Consider the following program. What is the output when the program is executed?

```
10    A$ = "APPLE"
20    B$ = "PINEAPPLE"
30    C$ = "ORANGE"
```

```
40    IF C$  > A$ AND C$ > B$ THEN PRINT C$ ELSE PRINT A$
50    END
```

A. APPLE
B. PINEAPPLE
C. ORANGE
D. ORANGE APPLE
E. ORANGE PINEAPPLE

28. Consider the following program. What is the output when the program is executed?

```
10    READ A, B, C
20    IF A > B THEN IF A > C THEN PRINT A ELSE PRINT C
                 ELSE IF B > C THEN PRINT B ELSE PRINT C

30    DATA  100,300, 200
40    END
```

A. 100
B. 300
C. 200
D. 100, 300, 200
E. None of the above

29. Consider the following program. What is the output when the program is executed?

```
10    A = 5
20    GOSUB 100
30    PRINT F
40    END
100   F = 1
110   FOR X = 1 TO A
120   F = F * X
130   NEXT
140   RETURN
```

A. 1
B. 5
C. 30
D. 120
E. None of the above

30. What is the output when the following program is executed?

```
100   READ T
110   X = T : GOSUB 200
120   X = Y : GOSUB 200
130   PRINT T, Y
140   DATA 5
150   END
200   Y = X * 2 + 1
210   RETURN
```

A. 5 13
B. 5 23
C. 11 23
D. 11 33
E. None of the above

Appendix 7

Test for the construction of program logic - BASIC group

TEST ON PROGRAMMING AND FLOWCHARTING (A)

NAME _____

Class : _____ Class no. : _____

Answer ALL questions.
Time allowed is 70 minutes.
Write all the draft work and the answers in the space provided.
Cross out any unwanted material before the end of the test

** You are NOT allowed to draw any flowchart.

A partial Character List for ASCII

Character	ASCII	Character	ASCII	Character	ASCII
0	48	I	73	b	98
1	49	J	74	c	99
2	50	K	75	d	100
3	51	L	76	e	101
4	52	M	77	f	102
5	53	N	78	g	103
6	54	O	79	h	104
7	55	P	80	i	105
8	56	Q	81	j	106
9	57	R	82	k	107
:	58	S	83	l	108
;	59	T	84	m	109
<	60	U	85	n	110
=	61	V	86	o	111
>	62	W	87	p	112
?	63	X	88	q	113
@	64	Y	89	r	114
A	65	Z	90	s	115
B	66	[	91	t	116
C	67	\	92	u	117
D	68	]	93	v	118
E	69	^	94	w	119
F	70	_	95	x	120
G	71	`	96	y	121
H	72	a	97	z	122

No. 1 (A1)

Write a program to count the number of different vowels in a sentence. A sentence(ST$), in capital letters only, is inputted. The numbers of different vowels should be outputted at the end of the program.

Input of the program : A sentence in capital letters only (ST$)

Output of the program : Number of "A" (NA),
Number of "E" (NE),
Number of "I" (NI),
Number of "O" (NO) and
Number of "U" (NU).

Sample input : "ALEX IS A LITTLE BOY AND HE LIVES IN KWUN TONG"

Sample output :
```
NUMBER OF A IS 3
NUMBER OF E IS 4
NUMBER OF I IS 4
NUMBER OF O IS 2
NUMBER OF U IS 1
```

****

Answer : (A1)

No. 2 (A2)

Write a program to find the number of matched pairs of numbers from two ordered lists of integers. The two lists of integers are inputted and there are 10 integers in each of the lists. After execution, the number of matched pairs should be outputted. It is assumed that there is no duplication of number in each of the lists.

Input of the program : Two lists of integers (LA and LB)

Output of the program : Number of matched pairs (NM)

Sample input : LA : 3, 5, 14, 34, 67, 68, 72, 74, 84, 87
LB : 5, 28, 37, 66, 67, 72, 79, 84, 100, 155

Sample output : Number of matched pairs = 4

****

Answer : (A2)

Appendix 8

Test for the construction of program logic - FLOWCHART group

TEST ON PROGRAMMING AND FLOWCHARTING (B)

NAME _____

Class : _____ Class no. : _____

Answer ALL questions.
Time allowed is 70 minutes.
Write all the draft work and the answers in the space provided.
Cross out any unwanted material before the end of the test

A partial Character List for ASCII

Character	ASCII	Character	ASCII	Character	ASCII
0	48	I	73	b	98
1	49	J	74	c	99
2	50	K	75	d	100
3	51	L	76	e	101
4	52	M	77	f	102
5	53	N	78	g	103
6	54	O	79	h	104
7	55	P	80	i	105
8	56	Q	81	j	106
9	57	R	82	k	107
:	58	S	83	l	108
;	59	T	84	m	109
<	60	U	85	n	110
=	61	V	86	o	111
>	62	W	87	p	112
?	63	X	88	q	113
@	64	Y	89	r	114
A	65	Z	90	s	115
B	66	[	91	t	116
C	67	\	92	u	117
D	68	]	93	v	118
E	69	^	94	w	119
F	70	_	95	x	120
G	71	`	96	y	121
H	72	a	97	z	122

No. 1 (B1)

Draw a detailed flowchart to count the number of different vowels in a sentence. A sentence(ST$), in capital letters only, is inputted. The numbers of different vowels should be outputted at the end of the detailed flowchart.

Input of the program : A sentence in capital letters only (ST$)

Output of the program : Number of "A" (NA),
Number of "E" (NE),
Number of "I" (NI),
Number of "O" (NO) and
Number of "U" (NU).

Sample input : "ALEX IS A LITTLE BOY AND HE LIVES IN KWUN TONG"

Sample output :
```
NUMBER OF A IS 3
NUMBER OF E IS 4
NUMBER OF I IS 4
NUMBER OF O IS 2
NUMBER OF U IS 1
```

****

Answer : (B1)

No. 2 (B2)

Draw a detailed flowchart to find the number of matched pairs of numbers from two ordered lists of integers. The two lists of integers are inputted and there are 10 integers in each of the lists. After execution, the number of matched pairs should be outputted. It is assumed that there is no duplication of number in each of the lists.

Input of the program : Two lists of integers (LA and LB)

Output of the program : Number of matched pairs (NM)

Sample input : LA : 3, 5, 14, 34, 67, 68, 72, 74, 84, 87
LB : 5, 28, 37, 66, 67, 72, 79, 84, 100, 155

Sample output : Number of matched pairs = 4

****

Answer : (B2)

Appendix 9

Test for the program composition skill - NON-FLOWCHART group

TEST ON PROGRAMMING AND FLOWCHARTING (C)

NAME _____

Class : _____ Class no. : _____

Answer ALL questions.
Time allowed is 70 minutes.
Write all the draft work and the answers in the space provided.
Cross out any unwanted material before the end of the test

** You are NOT allowed to draw any flowchart before writing the program.

A partial Character List for ASCII

Character	ASCII	Character	ASCII	Character	ASCII
0	48	I	73	b	98
1	49	J	74	c	99
2	50	K	75	d	100
3	51	L	76	e	101
4	52	M	77	f	102
5	53	N	78	g	103
6	54	O	79	h	104
7	55	P	80	i	105
8	56	Q	81	j	106
9	57	R	82	k	107
:	58	S	83	l	108
;	59	T	84	m	109
<	60	U	85	n	110
=	61	V	86	o	111
>	62	W	87	p	112
?	63	X	88	q	113
@	64	Y	89	r	114
A	65	Z	90	s	115
B	66	[	91	t	116
C	67	\	92	u	117
D	68	]	93	v	118
E	69	^	94	w	119
F	70	_	95	x	120
G	71	`	96	y	121
H	72	a	97	z	122

No. 1 (C1)

Write a program to find the highest average mark and the lowest average mark of 10 students. The Chinese mark(CM), The English mark(EM) and the Mathematics mark(MM) of each student are inputted. The highest average mark(HM) and the lowest average mark(LM) should be outputted at the end of the detailed flowchart. The output should be corrected to one decimal place.

Input of the program : Chinese mark (CM),
English mark (EM) and
Mathematics mark (MM)

Output of the program : The highest average mark (HM) and
The lowest average mark (LM)

Sample input :

CM	EM	MM
34	45	56
67	87	67
55	45	65
66	44	76
77	67	97
44	75	45
77	88	97
88	87	97
67	44	33
67	34	75

Sample output :
THE HIGHEST AVERAGE MARK IS 90.6
THE LOWEST AVERAGE MARK IS 45.0

****

No. 2 (C2)

Write a program which accepts a number(NUM) and determines whether it is a prime number or not. If it is a prime number, a sentence ("IT IS A PRIME NUMBER") should be printed, otherwise, the product of prime factors should be printed.

Input of the program : an integer (NUM)

Output of the program : "IT IS A PRIME NUMBER"
if it is a prime number
The product of prime factors
if it is not a prime number

Sample input(1) : 37

Sample output(1) : IT IS A PRIME NUMBER

Sample input(2) : 36

Sample output(2) : 2 x 2 x 3 x 3

****

Answer : (C2)

Appendix 10

Test for the program composition skill - FLOWCHART group

TEST ON PROGRAMMING AND FLOWCHARTING (D)

NAME _____

Class : _____ Class no. : _____

Answer ALL questions.
Time allowed is 70 minutes.
Write all the draft work and the answers in the space provided.
Cross out any unwanted material before the end of the test

A partial Character List for ASCII

Character	ASCII	Character	ASCII	Character	ASCII
0	48	I	73	b	98
1	49	J	74	c	99
2	50	K	75	d	100
3	51	L	76	e	101
4	52	M	77	f	102
5	53	N	78	g	103
6	54	O	79	h	104
7	55	P	80	i	105
8	56	Q	81	j	106
9	57	R	82	k	107
:	58	S	83	l	108
;	59	T	84	m	109
<	60	U	85	n	110
=	61	V	86	o	111
>	62	W	87	p	112
?	63	X	88	q	113
@	64	Y	89	r	114
A	65	Z	90	s	115
B	66	[	91	t	116
C	67	\	92	u	117
D	68	]	93	v	118
E	69	^	94	w	119
F	70	—	95	x	120
G	71	`	96	y	121
H	72	a	97	z	122

No. 1 (D1)

Draw a detailed flowchart and then write a program accordingly to find the highest average mark and the lowest average mark of 10 students. The Chinese mark(CM), The English mark(EM) and the Mathematics mark(MM) of each student are inputted. The highest average mark(HM) and the lowest average mark(LM) should be outputted at the end of the detailed flowchart. The output should be corrected to one decimal place.

Input of the program : Chinese mark (CM),
English mark (EM) and
Mathematics mark (MM)

Output of the program : The highest average mark (HM) and
The lowest average mark (LM)

Sample input :

CM	EM	MM
34	45	56
67	87	67
55	45	65
66	44	76
77	67	97
44	75	45
77	88	97
88	87	97
67	44	33
67	34	75

Sample output : THE HIGHEST AVERAGE MARK IS 90.6
THE LOWEST AVERAGE MARK IS 45.0

****

Answer : (D1)

No. 2 (D2)

Draw a detailed flowchart and then write a program accordingly which accepts a number(NUM) and determines whether it is a prime number or not. If it is a prime number, a sentence ("IT IS A PRIME NUMBER") should be printed, otherwise, the product of prime factors should be printed.

Input of the program : an integer (NUM)

Output of the program : "IT IS A PRIME NUMBER"
if it is a prime number
The product of prime factors
if it is not a prime number

Sample input(1) : 37

Sample output(1) : IT IS A PRIME NUMBER

Sample input(2) : 36

Sample output(2) : 2 x 2 x 3 x 3

****

Answer : (D2)

Appendix 11

Scoring sheet for Experiment I

Questions A1 and B1 :

Logic Elements

1	Accepts ST$
2	Initialization
3	Find number of character in ST$
4	Loop (counter 1 to LEN(ST$))
5	Extract a letter form ST$
6	Check the letter
7	Count vowels
8	Correct outputs after the loop
9	Logic element 1 and 2 before loop
10	Logic element 5, 6 and 7 within loop

* Each logic element carries one mark.

Questions A2 and B2

Logic Elements

1	Loop for reading list LA
2	Read list LA
3	Loop for reading list LB
4	Read list LB
5	Initialize NM
6	Loop - varying subscript of LA
7	Loop - varying subscript of LB
8	Test and count matched pairs
9	Print NM - outside nested loops
10	Logic element 8 inside the inner loop

* Each logic element carries one mark.

Appendix 12

Scoring sheet for Experiment II

Question C1 and D1 :

Logic Element

1	Initialization
2	Loop (counter 1 to 10)
3	Accepts CM, EM, MM
4	Calculate average
5	Correct average to one decimal place
6	Find highest
7	Find lowest
8	Print result after the loop
9	Logic element 1 before the loop
10	Logic elements 3, 4, 5, 6 and 7 within loop

* Each logic element carries one mark.

Question C2 and D2 :

Logic Elements

1	Accepts NUM
2	Set Flag
3	Loop (counter 2 to NUM/2 or sq.root of NUM)
4	Test if NUM is divisible by counter (X)
5	Correct action(set FLAG and print X) if yes
6	Correct action(increase loop counter)if no
7	Test Flag
8	Correct message when FLAG = 0
9	Print NUM when FLAG = 1
10	Logic elements 4, 5, 6 and within Loop

* Each logic element carries one mark.

Appendix 13

Questionnaire for the subjects

QUESTIONNAIRE

Please fill in the following information before the test.

1. Name : _____(English - in block letters)

   Class : _____ Class no. : _____

2. Sex (M/F) : ____

3. Age : ____

4. Did you study Computer Literacy in
   a. F. 1 ? _____(yes/no)
   b. F. 2 ? _____(yes/no)
   c. F. 3 ? _____(yes/no)

5. Do you have a home computer ? _____(yes/no)

   If yes, please answer the following question.
   (Give a tick for the right answer)

   a. What is the major purpose for you to use the home computer ?
      i. For playing computer game _____
      ii. For word processing _____
      iii. For programming _____
      iv. For other purpose _____
      (please specify : _____)

   b. How much time do you spend on the home computer in one week?
      i. Less than 2 hours _____
      ii. About 2 to 5 hours _____
      iii. More than 5 hours _____

6. Had you attended other computer course outside the school?
   _____(yes/no)

   If yes, please answer the following questions:
   (Give a tick for the right answer)

   a. What is the content of the computer course?
      i. Programming in BASIC _____
      ii. Programming in other language _____
      iii. Basic computer concept _____
      iv. Word processing _____
      v. Others _____
      Please specify : _____

b. What is the duration of the course ?

i. less than 10 hours _____

ii. about 10 to 20 hours _____

iii. more than 20 hours _____

7. Did you learn flowcharting in Form 4 ? _____(yes/no)

For questions 8 to 11, give a circle on the horizontal scale.

8. Do you draw flowcharts to organize the program logic before writing the program?

Never Very often

1 2 3 4 5

1____1____1____1____1

9. Do you agree that flowcharting helps you to compose a program?

Not agree Agree

1 2 3 4 5

1____1____1____1____1

10. Do you agree that flowcharting is a more difficult task than writing program?

Not agree Agree

1 2 3 4 5

1____1____1____1____1

11. When you are required to submit a program with a flowchart, do you write the program first and then draw the flowchart accordingly?

Never Very often

1 2 3 4 5

1____1____1____1____1

** End of questionnaire **